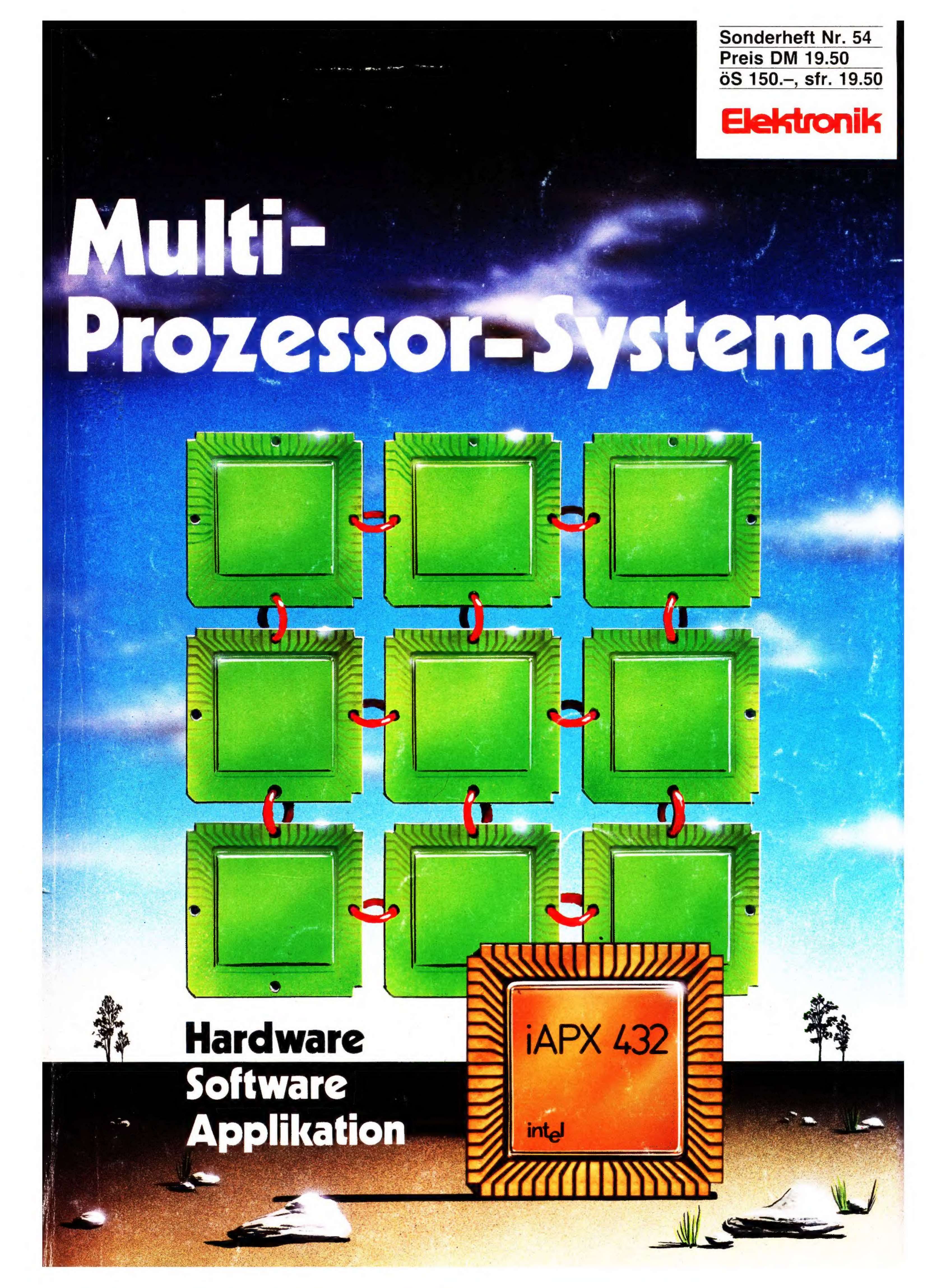


Sonderheft Nr. 54
Preis DM 19.50
öS 150.–, sfr. 19.50
Elektronik
Multi-Prozessor-Systeme
Hardware
Software
Applikation
iAPX 432
intel

Vorwort

Komplexere Aufgaben erfordern leistungsfähigere Lösungen. Diese allgemeingültige Aussage trifft wie für alle Lebensbereiche ganz besonders auch für die Elektronik zu.

Auf die Mikroprozessortechnik übertragen heißt das, daß den wachsenden Anforderungen an Systeme auch entsprechend leistungsfähige Konzepte entgegengestellt werden müssen.

Wie steigert man die Leistung von Mikrocomputersystemen? Schnellere Verarbeitung und damit höherer Durchsatz bedeutet Verwendung von Hochgeschwindigkeits-Bauelementen, die teuer und nicht einfach zu handhaben sind. Ein breites Datenwort, also der Schritt vom 16- zum 32- oder sogar 64-Bit-Prozessor, bedeutet eine Erhöhung der IC-Komplexität, die ebenfalls nicht unproblematisch ist.

Die Lösung, bei der heute verfügbare Techniken und Produkte Verwendung finden, liegt im Multi-Mikrocomputer-Konzept.

Systeme mit mehreren Prozessoren sind natürlich auch nicht leicht zu konzipieren und realisieren. Hilfestellung sowie Anregungen dazu bieten die in dem vorliegenden Heft zusammengefaßten Beiträge.

Die Redaktion

Inhalt

(Titelbild: Intel)

1982
Franzis-Verlag GmbH, Karlstraße 37–41, 8000 München 2.
Produktion: ELVAG, Elektronik Verlag Luzern AG, CH-6002 Luzern.
Bearbeitet von der Redaktion der Zeitschrift ELEKTRONIK.
Für den Text verantwortlich: Peter von Bechen.

ISSN 0170-0898
Druck: Stämpfli + Cie AG, CH-3001 Bern
ZV-Artikel-Nr. 54021 · ST/ZV/1082/673/7'

Hermann Schmid

Multi-Mikroprozessor-Systeme

1. Teil: Grundlagen

Konfigurationen mit mehreren Mikroprozessoren haben seit einiger Zeit große Bedeutung erlangt. In vielen Anwendungsbereichen sind Konfigurationen, bei denen die Verarbeitungsleistung auf mehrere Prozessoren verteilt ist, Konzepten mit nur einem leistungsfähigen Zentralrechner überlegen. Vorteile sind beispielsweise hohe Flexibilität und geringere Störanfälligkeit. Die ELEKTRONIK hat in den letzten Jahren schon häufig über Mehrprozessorsysteme berichtet und die Konzepte einzelner praktisch ausgeführter Systemlösungen beschrieben. In der mit diesem Beitrag beginnenden Aufsatzreihe soll die Technik der Multi-Mikroprozessor-Systeme umfassend und systematisch dargestellt werden. Moderne Entwicklungen, beispielsweise die kürzlich vorgestellten 32-Bit-Prozessoren, werden auch berücksichtigt. Der erste Teil der Beitragsreihe befaßt sich mit den Grundlagen der Mehrprozessorsysteme. Es werden insbesondere die Begriffe definiert und der Zusammenhang zwischen ihnen dargestellt.

1 Warum Multi-Mikroprozessor-Systeme?

Kleine, preiswerte, aber trotzdem leistungsfähige Digitalrechner bieten heute Lösungsmöglichkeiten für viele Probleme:

- Energieeinsparung im Haushalt, in Fabrikationsstätten, Automobilen usw.
- Produktionseinrichtungen mit höherer Effizienz, z. B. für Lebensmittel, Textilien usw.
- Systeme für Kommunikation, Ausbildung, Unterhaltung usw.

Eine Möglichkeit, die Leistung von Computern zu steigern, ist, deren Arbeitsgeschwindigkeit zu erhöhen. Allerdings erfordern schnelle Schaltelemente, beispielsweise in ECL-Technik aufgebaut, höhere Betriebsleistung und eignen sich zudem nicht zur Integration in LSI-Strukturen. Die Implementierung eines typischen Minicomputers mit Bauelementen mittlerer Integrationsdichte erfordert nicht nur ein breites Spektrum unterschiedlicher Typen, sondern auch viele Platinen und einige tausend elektrische Verbindungen (*Bild 1*).

Dagegen kann man bei einem Multi-Mikroprozessor-System (MMPS) hohe Leistung bei Verwendung identischer LSI-Bausteine erreichen. Es handelt sich dabei jeweils um komplette Prozessoren mit CPU, Speicher sowie E/A-Einheit, die mit relativ geringer Taktfrequenz arbeiten. Diese Prozessoren können untereinander in einem einfachen Leitungspaar verbunden werden, wodurch der Aufwand sehr gering ist. Außerdem benötigt ein MMPS nur eine geringe Anzahl Platinen und Stecker. Daraus ergibt sich die sehr hohe Hardware-Ökonomie eines MMPS.

Das MMPS bietet außerdem architektonische Flexibilität und Erweiterbarkeit. Mit einer einzigen Architektur (die aber sorgfältig konzipiert sein sollte) kann eine große Anzahl von Systemen implementiert werden, unabhängig von den jeweiligen Anforderungen an Durchsatz, Speicher- oder E/A-Kapazität.

Zukünftig werden an erster Stelle die Anforderungen für digitale Prozessor-Hardware die Zuverlässigkeit, Wartbarkeit, Überlebensfähigkeit, Sicherheit sowie geringe Kosten während des Lebenszyklus sein. Erfüllt werden diese Forderungen von fehlertoleranten Mehrprozessorsystemen.

Wartbarkeit ist die Fähigkeit, Fehler zu entdecken, zu isolieren und zu beseitigen, so daß das System nach einer Fehlfunktion weiterarbeiten kann. Der Aufwand für die Wartung von technischen Einrichtungen (beispielsweise Automobilen, Flugzeugen, Computern usw.) hat sich zu einem signifikanten Prozentsatz der Gesamtkosten entwickelt. Zukünftige Geräte müssen in der Lage sein, auch nach Auftreten von Fehlern zu funktionieren. Mit der Verfügbarkeit von preiswerter digitaler Hardware fallen die Kosten für redundante Prozessoren im Vergleich zu Reparatur- und Ausfallkosten nicht mehr ins Gewicht.

Es gibt einige Gründe, warum MMPS noch nicht so häufig verwendet werden:

- Einchip-Prozessoren mit ausreichender CPU-, Speicher- und E/A-Leistung werden jetzt erst verfügbar.

- Bis jetzt gibt es nur wenige erprobte Techniken für die Kommunikation, Prioritätsdecodierung und Arbitration zwischen den Prozessoren.
- Methoden, Werkzeuge und Betriebssysteme für die Softwareentwicklung sind noch nicht ausgereift und daher kommerziell noch nicht verfügbar.

2 Welche Multi-Mikroprozessor-Systeme gibt es?

Ein MMPS soll hier als ein Verarbeitungssystem definiert werden, welches:

- mindestens zwei µPs, die jeweils aus einer CPU, einem lokalen Speicher sowie lokalen E/A-Einheiten besteht;
- die Möglichkeit bietet, daß alle Einzelprozessoren miteinander kommunizieren können, indem sie auf einen Globalspeicher (der zentral oder verteilt angeordnet sein kann) zugreifen;
- die Möglichkeit zum Datenaustausch zwischen allen Prozessoren über eine oder mehrerere System- oder Globalbusse bietet;
- alle Prozessoren am gleichen physikalischen Ort zusammenfaßt.

Viele Multiprozessorkonfigurationen wurden bereits entwickelt, allerdings bildeten die meisten Mini- und Großcomputer [1]. *Anderson* [2] hat die verschiedenen Architekturen folgendermaßen klassifiziert:

1. SIMD-Systeme (*Single Instruction, Multiple Data*), z. B.:
   - Array-Prozessoren
   - Assoziative Prozessoren
2. MIND-Systeme (*Multiple Instruction, Multiple Data*), z. B.:
   - Verteilte Multiprozessorsysteme (geographisch verteilt).
   - Zusammengefaßte Multiprozessorsysteme (am gleichen Ort).

Zusammengefaßte Multiprozessorsysteme lassen sich wiederum folgendermaßen einteilen:

a) Mehrere CPUs teilen sich einen gemeinsamen Speicher (*Bild 2a*). Diese Version ist sehr uneffizient und wird daher nicht näher erläutert.

b) Mehrere Prozessoren (CPU, Speicher und E/A) an einem gemeinsamen Bus (Bild 2b). Diese Version ist am leichtesten realisierbar und wird daher heute sehr oft verwendet.

c) Hierarchische Systeme in drei Ebenen (Bild 2c). Diese Version kann als Erweiterung von b) zur Realisierung größerer Systeme angesehen werden.

d) Prozessor- oder Kreuzschienen-Netzwerke, mit denen N Verarbeitungsmodule mit N Speichermodulen über Kreuzschienenschalter verbunden sind (Bild 2d). Diese Version eignet sich für die Implementierung von Systemen mit Großrechnerleistung.

Theoretisch läßt sich jedes zusammengefaßte MMPS zu einem fehlertoleranten Verarbeitungssystem machen, indem eine dynamische Rekonfiguration hinzugefügt wird, nämlich die Möglichkeit:

- Fehler zu entdecken und zu isolieren, die Beschädigung einzuschätzen und diese der Steuereinheit mitzuteilen;
- den fehlerhaften Prozessor durch einen Ersatz im dynamisch redundanten System auszutauschen;
- die Tasks (Software-Prozesse) in einer vorherbestimmten Weise neu zu verteilen;
- der direkten Neuverteilung durch zentrale oder verteilte Steuereinheit.

Obwohl die Notwendigkeit für fehlertolerante Computer allgemein anerkannt wird [3] und dies immer als endgültiges Entwicklungsziel für Multiprozessor-Konzepte sein sollte, beschäftigt sich dieser Beitrag nur mit Systemen, bei denen jeder Prozessor einer speziellen Aufgabe zugeteilt ist. Außerdem sollen hier nur solche Systeme beschrieben werden, bei denen komplette Prozessoren über einen parallelen Systembus miteinander kommunizieren, wie das in Bild 2b dargestellt ist.

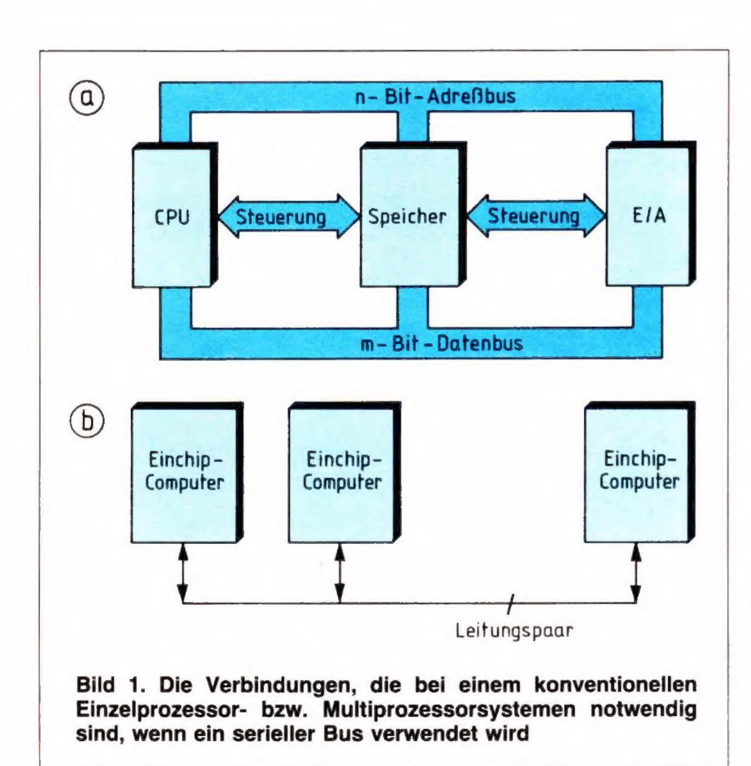


Bild 1. Die Verbindungen, die bei einem konventionellen Einzelprozessor- bzw. Multiprozessorsystemen notwendig sind, wenn ein serieller Bus verwendet wird

3 Multi-Mikroprozessor-Systembusse

Der Systembus ist das Rückgrat eines jeden Prozessorsystemes. Durch diesen ist nicht nur die Hardware, Software und die Kommunikationsprotokolle zur Verknüpfung der Prozessoren oder der gemeinsamen Ressourcen mit dem Bus definiert, sondern auch standardisiert. Es gibt viele Buskonfigurationen, z. B. seriell/parallel, einfach/redundant usw., wobei jede Lösung einen Kompromiß zwischen Leistung, Kosten sowie dem Grad der Verknüpfung zwischen den Prozessoren darstellt. Allgemein kann man sagen, daß serielle Busse geringe Kosten verursachen (es ist nur ein Leitungspaar

erforderlich), eine lose Kopplung und eine geringe Datentransferrate bieten. Daher werden sie vornehmlich für die Verbindung über längere Distanzen (>1 m) in verteilten Systemen benutzt. Im Gegensatz dazu haben Parallelbusse mit mehreren Leitungen eine festere Kopplung, ermöglichen höhere Datenraten, sind aber teurer. Serielle Busse erfordern wesentlich aufwendigere Kommunikationsprotokolle, beispielsweise HDLC oder X.25, die an anderer Stelle dieser Reihe beschrieben werden. Hier sollen nur Systeme mit parallelen Bussen, wie Multibus, Z-Bus oder S-100-Bus, behandelt werden.

Die Parallelbusse für Systeme mit einem oder mehreren Prozessoren sind normalerweise in einer Party-Line-Konfiguration implementiert, indem alle Steckanschlüsse auf der Verdrahtungsrückwand des Computers parallel miteinander verbunden werden. Die Gesamtlänge der Leitungen und ihre charakteristische Impedanz bestimmen die Grenzen der Übertragungsrate und Transportverzögerungen. Derzeit verwendete Mikroprozessorsystem-Busse, wie Multibus, Z-Bus oder S-100, haben zwischen 50 und 100 Leitungen.

Adressen- und Datenleitungen werden manchmal im Multiplex geschaltet, wie das beim Z-Bus der Fall ist. Anderenfalls besitzt jedes Bit oder jede Stromzuführung eine eigene Leitung. Die Implementierung des Steuerbereiches eines Busses ist problematischer, weil dadurch die Betriebsweise des Gesamtsystems bestimmt wird. Die Entscheidung, welche Steuerleitungen benutzt werden sollen, wie sie zusammenzufassen, zu codieren, zu multiplexen und implementieren sind, ist eine Aufgabe, die außerordentlich viel Erfahrung erfordert, insbesondere wenn das System Multiprozessor-Betrieb unterstützen muß.

Ein Systembus für ein Multiprozessor-Steuersystem muß zusätzlich zum konventionellen Datentransfer und den Steuerleitungen Einrichtungen zur Steuerung folgender Funktionen enthalten:

- Anforderung des Busses oder einer anderen gemeinsamen Ressource;
- Decodierung der Priorität aller Partner (bis zu 64);
- Arbitrierung der Anforderungen mit hoher Geschwindigkeit;
- Übergabe der Steuerung an neuen Bus-Master (Bus-Grant);
- Adressierung einer jeden Datenquelle im System;
- Rekonfiguration des Systems oder Wiederaufnahme der Erledigung von Tasks;
- Überwachung und Test aller Busoperationen.

Die meisten vorhandenen Mikroprozessor-Systembusse bieten nur sehr begrenzte Möglichkeiten zur Multiprozessor-Unterstützung. Der Multibus unterstützt beispielsweise nur die Daisy-Chain-Prioritätsdecodierung/Arbitration von bis zu vier Partnern sowie nur einfache Bus-Request/Grant-Protokolle. Dagegen umfaßt der neue Backplane-Busstandard, der von dem IEEE Computer Society Microprocessor Standard Committee (P896) vorbereitet wird, die meisten der oben aufgeführten Punkte.

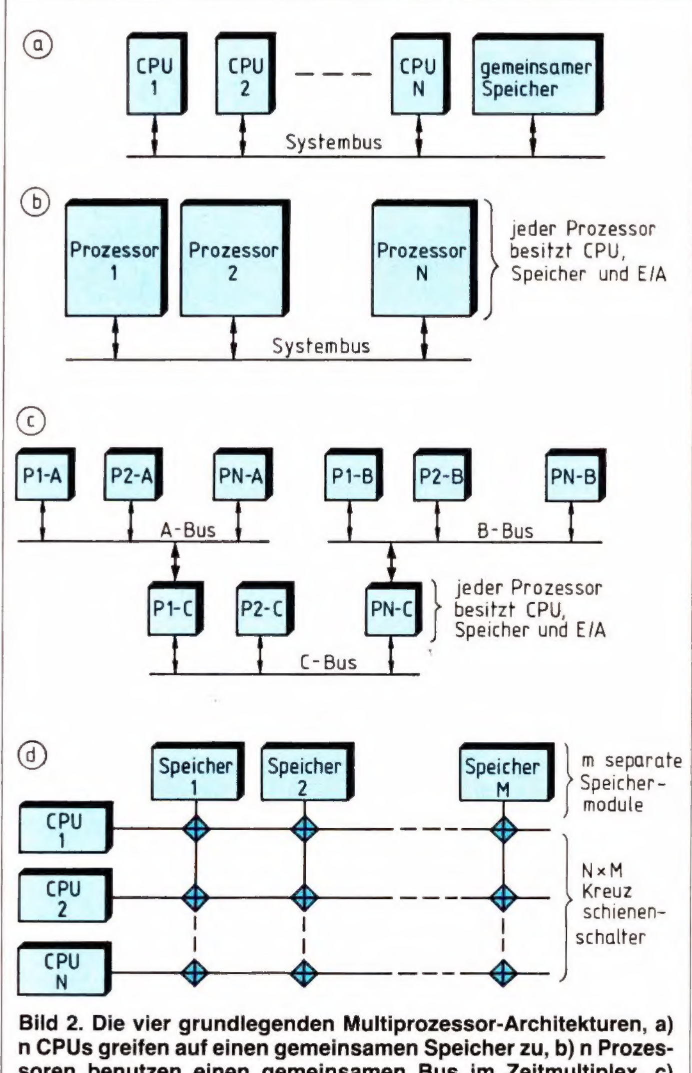


Bild 2. Die vier grundlegenden Multiprozessor-Architekturen, a) n CPUs greifen auf einen gemeinsamen Speicher zu, b) n Prozessoren benutzen einen gemeinsamen Bus im Zeitmultiplex, c) hierarchisches System auf drei Ebenen, d) Kreuzschienen-Netzwerk

4 Speicherkonfigurationen

Typ, Größe und Lage des Speichers haben großen Einfluß auf die Eigenschaften des MMPS. Der Gesichtspunkt bei der Auswahl und der Lokalisierung des Speichers muß immer so sein, daß die Datenmenge, die zwischen den Prozessoren über den Systembus transferiert werden muß, möglichst klein ist. Die wichtigsten Fragen sind dabei:

- Lokaler oder gemeinsamer Speicher? Oder beides?
- Wieviel lokaler Speicher, welcher Typ und welche Zugriffsarten?
- Wie sollen die Speicher für einfache Adressierung organisiert sein?

Alle diese Erwägungen repräsentieren architektonische Gesichtspunkte, die unter Berücksichtigung moderner Hardware- und Softwareentwicklungen gesehen werden müssen. Beispielsweise ist ein lokaler Speicher nicht mehr so kostenintensiv, wie es vor nicht allzu langer Zeit war, weil mittlerweile preiswerte monolithi-

sche Prozessoren zur Verfügung stehen, die auf dem Chip einen Speicher besitzen.

4.1 Gemeinsamer Speicher für gemeinsame Variable

In früheren Multiprozessorsystemen (Bild 2a) teilten sich mehrere CPUs einen Speicher in dem die lokalen und globalen Variablen abgelegt waren. Der Speicher wurde damit zur Engpaßstelle des Gesamtsytems. Neuerdings speichert man nur die gemeinsamen Variablen in gemeinsamen Speichern. Dieser Speicher übernimmt eine Funktion wie ein öffentliches „Schwarzes Brett", durch das jeder Zugang zu allen Informationen hat. Zugriffskonflikte sind weitgehend ausgeschlossen, weil keine lokalen Variablen gespeichert sind.

4.2 Lokaler Speicher für lokale Variable

Jede CPU muß ihren eigenen Speicher zum Ablegen von Anwenderprogrammen und lokalen Variablen haben, so daß sich über eine Instruktion oder lokale Variable unabhängig vom Systembus oder dem gemeinsamen Speicher verfügen kann. Das führt dazu, daß nicht nur die Operationen der CPU schneller sind, sondern beläßt den Systembus und den gemeinsamen Speicher der Kommunikation zwischen den Prozessoren. Die Menge von Daten, die im gemeinsamen Speicher abgelegt ist und über den Systembus transferiert werden muß, ist stark reduziert.

Der Zugriff auf lokale Speicher vom Systembus kann folgendermaßen erfolgen:

- Über Interrupt-Anfrage an die lokale CPU, die den Systembus an den lokalen Bus nach Ausführung der laufenden Instruktion anbindet. Die anfragende CPU muß während dieser Zeit warten, was eine Verzögerung von einigen 100 µs bei längeren Multiplikations- oder Divisions-Instruktionen bedeutet. Außerdem ist, während die andere CPU auf den Speicher zugreift, die lokale CPU untätig.
- Über direkten Speicherzugriff oder Busaufruf der lokalen CPU, die auch den Systembus mit dem lokalen Bus verbindet, allerdings in diesem Fall schneller. Die anfragende CPU muß nur warten, bis die lokale CPU den gerade laufenden Maschinenzyklus beendet hat, der bei den Hochleistungs-µPs nur einige µs beträgt (2,5 µs beim Z8000). Trotzdem ist die lokale CPU untätig, während die externe CPU auf den lokalen Speicher zugreift.
- Direkt und ohne Verzögerung auf RAMs mit zwei Ports. Weder die anfragende noch die lokale CPU verschwenden Zeit, in der sie aufeinander warten müssen. Das ist die ideale Lösung.

4.3 Lokale „Mailboxes" für die Kommunikation

Ein gemeinsamer Speicherbereich kann entfallen, wenn jeder Prozessor eine lokale „Mailbox" besitzt (*Bild 3*), die einem individuellen „Schwarzen Brett" für die Mitteilung allgemein verfügbarer Variablen für die anderen Partner entspricht. Ausschließlich die lokale CPU kann in ihre Mailbox schreiben, alle anderen können nur daraus lesen. Dadurch wird auch verhindert, daß fehlerhaft arbeitende Prozessoren die Daten in der Mailbox zerstören können oder den Bus durch dauernde Schreibaktionen verstopfen. Die lokale Mailbox hat einen weiteren Vorteil. Falls sie ausfällt, ist nur ein Teil des Systems unbenutzbar. Dagegen fällt das gesamte System aus, wenn eine globale Mailbox versagt.

4.4 Dual-Port-RAM

Die Eigenschaften des Gesamtsystems verbessern sich erheblich, wenn die lokale Mailbox mit einem Dual-Port-RAM implementiert ist.

In diesem Fall kann die lokale CPU in die Mailbox schreiben, während zur gleichen Zeit eine andere CPU aus dieser lesen kann. Aus diesem Grund muß die andere CPU nicht auf den Zugriff zur Mailbox warten.

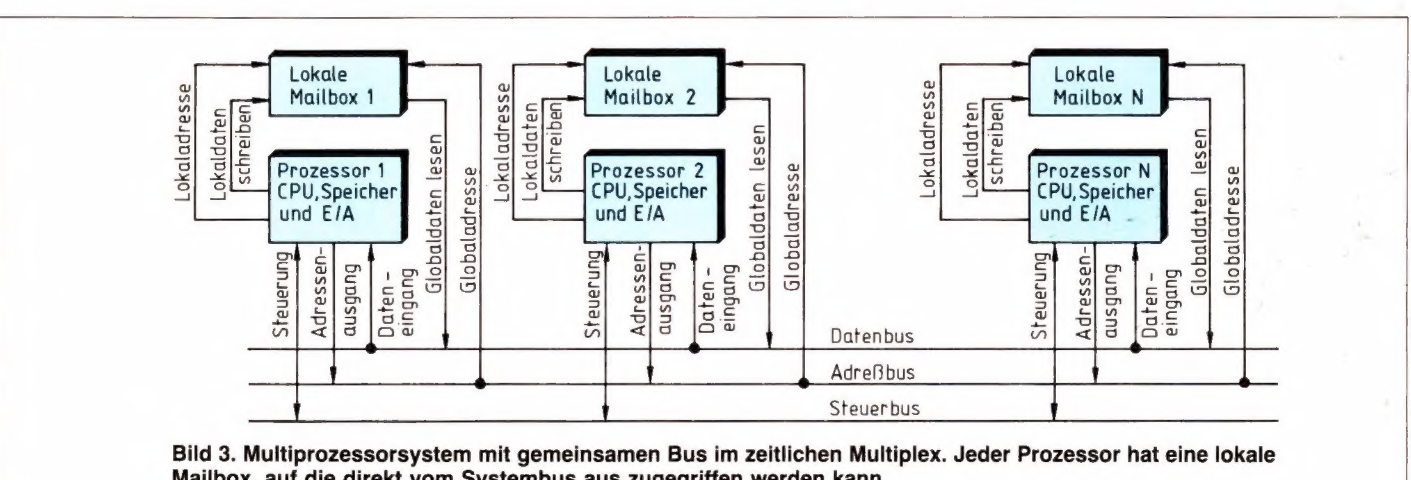


Bild 3. Multiprozessorsystem mit gemeinsamen Bus im zeitlichen Multiplex. Jeder Prozessor hat eine lokale Mailbox, auf die direkt vom Systembus aus zugegriffen werden kann

Der Einplatinen-Computer SBC-86/12 von Intel [4] bietet ein Quasi-Dual-Port-RAM (Adressen- und Datenleitungen werden zwischen zwei Benutzern umgeschaltet). Die ersten echten Dual-Port-RAMs mit nennenswerter Größe sind mittlerweile kommerziell verfügbar. Der Baustein MC68340 von Motorola ist seit 1982 erhältlich [5].

4.5 *Memory-Mapping*

Damit jede Mailbox für alle CPUs im System leicht erreichbar ist, müssen die lokalen Speicher eines jeden Prozessors innerhalb eines globalen Speicherbereiches gelegt werden. Als Beispiel soll das in *Bild 4* dargestellte System dieses Verfahren zeigen. Der 64-KByte-Speicher eines jeden der acht Prozessoren wurde aus dem 512-KByte-Speicher zugewiesen. Eine 3-Bit-Adresse (A21...A23) wählt das gewünschte Segment aus. Jeder 64-K-Speicherbereich ist in folgende Segmente aufgeteilt: der Programm-, der lokale Variablen- und der globale Variablenbereich, wobei letzterer die Mailbox darstellt. Nur die lokale CPU kann dort hineinschreiben. Um die Mailbox eines anderen Prozessors zu adressieren, muß die lesende CPU die drei höherwertigen Bits zu ihrer lokalen Adresse hinzufügen.

5 Operationen des Multiprozessorsystems

Ein Multiprozessorsystem unterscheidet sich von einer Anordnung mehrerer einzelner Prozessoren wie ein Arbeitsteam von einer Gruppe individueller Arbeiter. Das Ziel eines Arbeitsteams ist die Ausführung der Gesamtaufgabe innerhalb kürzester Zeit, der Gesichtspunkt für einen individuellen Arbeiter ist die Fertigstellung seiner spezifischen Aufgabe mit geringstem Zeitaufwand. Die Benutzung von einzelnen Arbeitern oder Prozessoren ist daher ineffizient, insbesondere wenn der Umfang der Aufgaben stark unterschiedlich ist, oder wenn eine Aufgabenlösung das Resultat oder Teillösungen anderer Aufgaben erfordert. Im Gegensatz dazu arbeiten Teams oder Multiprozessorsysteme effizient, weil deren einzelne Mitglieder miteinander kommunizieren, kooperieren und die Arbeitsbelastung aufteilen. Im einzelnen benötigt ein Multiprozessorsystem daher:

- Einen Systembus zum Transfer von Adressen-, Daten- und Steuerinformationen zwischen jeweils zwei Prozessoren
- Prioritäts-Decodierungs/Arbitrations-Hard- und -Software zur Steuerung des Zugriffs zu allgemeinen Ressourcen wie Speicher, E/A-Einheiten, Systembus usw.
- Ein Betriebssystem zur Steuerung, Verwaltung und zum zeitlich planmäßigen Ablauf von Verarbeitungsaufgaben sowie der Benutzung der Ressourcen. Außerdem muß festgelegt werden, wer wen zu welchem Zeitpunkt steuert.

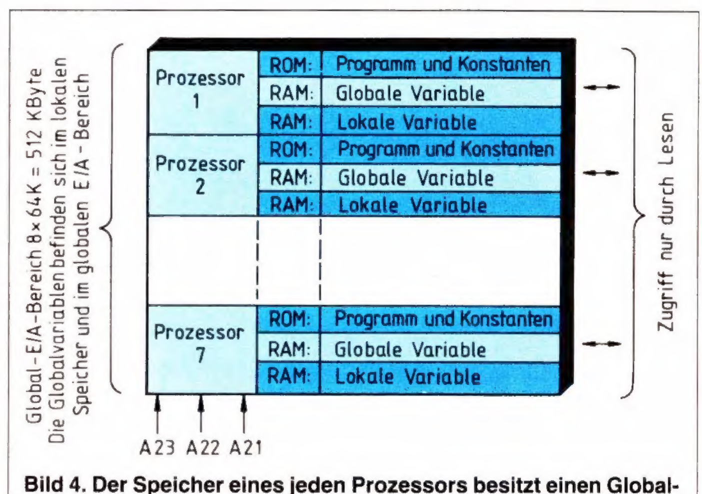


Bild 4. Der Speicher eines jeden Prozessors besitzt einen Global- bzw. E/A-Bereich

5.1 *Datentransfer zwischen Prozessoren*

Der Grund einer jeden Kommunikation zwischen Prozessoren ist der Austausch von Daten. Die Transferoperation belastet sowohl die Datenquelle als auch den Empfänger. Die Zeit, die für die Transferoperation erforderlich ist, gehört zum „Overhead" und reduziert die Gesamteffizienz des Systems. Aus diesem Grund sind für ein Hochleistungs-Multiprozessorsystem eine möglichst hohe Transferrate und geringe Verzögerungszeiten erforderlich.

Daten können über einen parallelen Bus mit einfachen Schreib-/Lese-E/A-Operationen und vier Schritten transferiert werden:

- Einleitung des Transfers mit Hard- oder Software;
- Auswahl von Quelle/Empfänger durch die entsprechenden Adressierungssignale;
- Abfrage von Quelle/Empfänger zur Ein-/Ausgabe der Daten von/auf den Bus;
- Weiterführung des Prozesses.

Die Datenrate (Wörter/s) und die Anzahl der Datenleitungen legt die Datentransferkapazität des Systembusses fest und damit den Schlüsselfaktor, der die Leistung des Gesamtsystems bestimmt. Die MMPS-Leistung wird außerdem durch die Bus-Zugriffszeit, die Anwahlverzögerung eines Partners, die Antwortzeit eines Partners usw. beeinflußt. Alle diese Faktoren werden im dritten Beitrag dieser Reihe näher diskutiert.

5.2 *Transfer der Steuerung zwischen den Prozessoren*

Bei dem MMPS, der in Bild 2b dargestellt ist, muß jeder Prozessor mit jedem anderen über den Systembus kommunizieren. Allerdings kann nur ein Prozessor den Systembus steuern. Daraus folgt, daß die verschiedenen Prozessoren beim Zugriff zum Systembus in Konkurrenz stehen. Diese Wettbewerbssituation muß durch Zuweisung in vorher festgelegte Prioritäten durch einen Arbi-

ter gelöst werden. Bevorzugt bei Zuweisung ist der jeweils „derzeitige Master". Um einen Prozessor zum Bus-Master zu machen, ist es erforderlich, daß er die Steuerung des Busses übertragen bekommt. Dafür sind die folgenden Schritte erforderlich:

- Aufstellung von Prioritäten durch
  - räumliche Decodierung (*Daisy Chain*),
  - Zuweisung eines Prioritätcodes für jeden Prozessor;
  - umlaufende Priorität (der Prozessor, der als letzter die Steuerung übernommen hat, bekommt die geringste Priorität);
- Initialisierung von Bus-Anfragen (eine Sequenz von Mikroprogramm-Schritten), um
  - zu überprüfen, ob der Systembus gerade benutzt wird und die Anfrage abzubrechen, wenn dies der Fall ist;
  - die Bus-Request-Leitung zu aktivieren, damit die Anfrage allen anderen signalisiert wird,
  - zu warten, daß die Anfrage entschieden, bewilligt oder zurückgewiesen wird;
- Entscheidung über Anfragen lokal oder zentral durch
  - Identifizierung des Anfragenden über seine räumliche Position oder den Code (Vektor),
  - Vergleich des seriellen oder parallelen Prioritäts-Codes des Anfragers,
  - Freigabe der Einheit mit der höchsten Priorität und Sperren aller anderen durch Aussenden eines Bus-Bewilligungssignals zum neuen Master.

Der neue Master kann nun den Systembus steuern. Allerdings ist es das Ziel, daß Daten zwischen dem Master und anderen globalen Ressourcen (Speicher usw.) transferiert werden. Deshalb muß der Partner zu seinem lokalen Bus Zugriff erlauben. Wenn die globalen Variablen in einem Dual-Port-RAM gespeichert sind, ist dieser Zugriff dauernd erlaubt. Der neue Master erzeugt ganz einfach die entsprechende Globaladresse und führt eine E/A-Operation aus. In derzeit üblichen Multiprozessorsystemen werden Interrupt-, DMA- oder durch Bus-Request gesteuerter Zugriff auf den lokalen Spei-

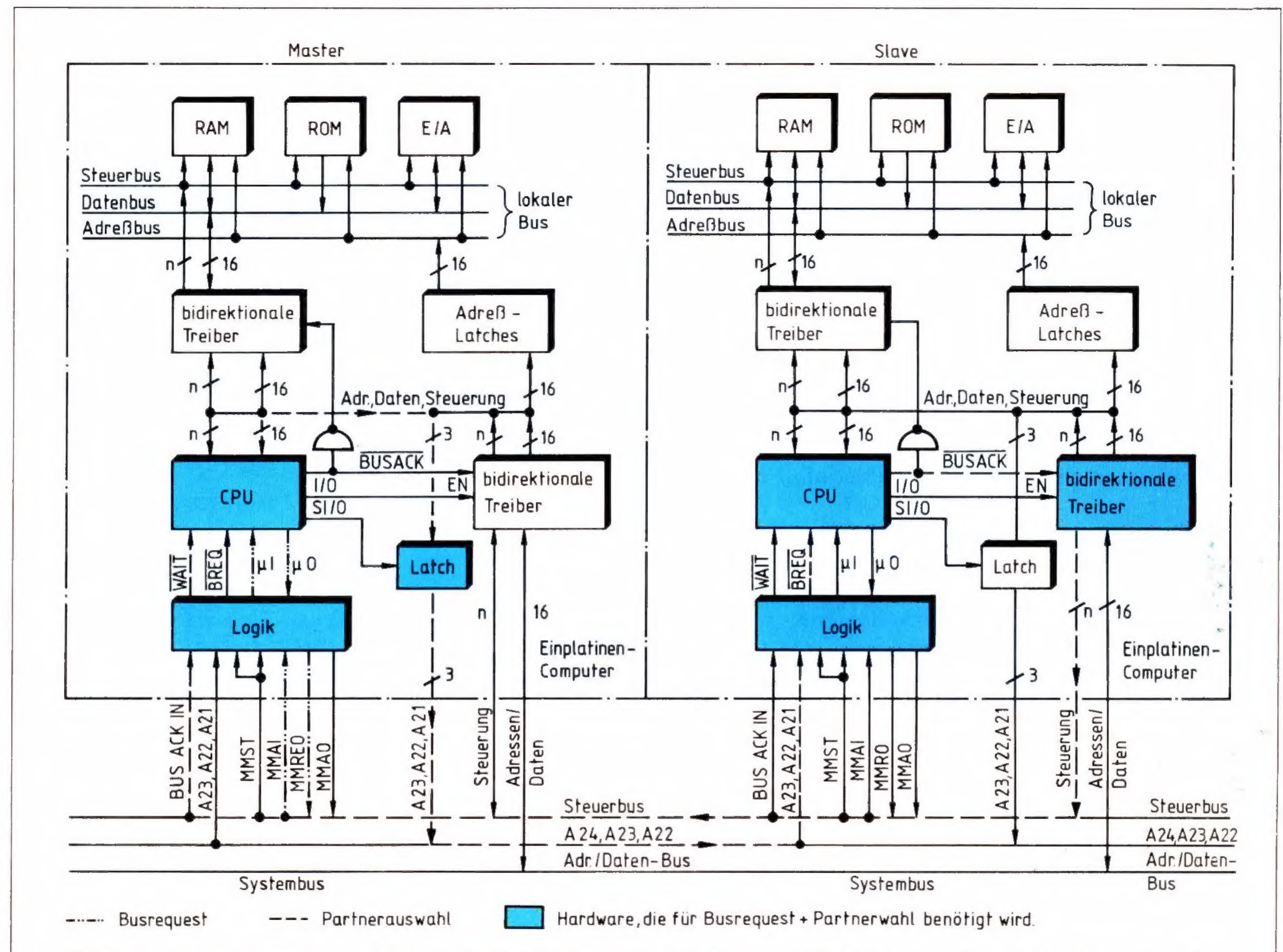


Bild 5. Nach einer Busanfrage wird der linke Prozessor Busmaster. Danach fordert er den Zugriff auf den lokalen Bus des rechten Prozessors durch Ausgabe einer 3-Bit-Adresse. Nur wenn dieser die Steuerung des Busses übergibt (BUSACK), können die Daten transferiert werden

cher benutzt, weil Dual-Port-RAMs noch nicht in genügender Zahl verfügbar sind.

Das heißt, daß der Zugriff auf einen lokalen Speicher die Übernahme der Steuerung der lokalen Ressourcen des Partners bedeutet. Dazu sind die folgenden Schritte erforderlich (*Bild* 5):

- Auswahl des Partners durch Ausgabe der Adresse (3 Bit), die wiederum einen Interrupt, DMA- oder Bus-Request für die CPU des Partners erzeugt.
- Start eines Zählers, für die Zeitbegrenzung, wenn der Partner den Zugriff nicht in einer vorher festgelegten Periode bewilligt.
- Der Partner führt die laufende Instruktion bzw. den Maschinenzyklus aus, schaltet sich vom lokalen Bus ab und gibt danach das Busfreigabe-Signal aus.
- Das Busfreigabe-Signal gibt dem neuen Master die Steuerung der lokalen Ressourcen des Partners und erlaubt die gewünschte Datentransfer-Operationen.

Die Entscheidung, welche dieser Funktionen in der Hardware oder in der Software implementiert sein sollen, ob die verschiedenen Operationen lokal vom Prozessor oder zentral von einer System-Steuereinheit ausgeführt werden sollen, hängt von den Entwicklungsrichtlinien ab, die durch die Anwendung sowie die verfügbare Hard- oder Software beeinflußt werden.

6 Implementierung des Prioritäts-Decodierers/Arbiters

Die Zuweisung von Prioritäten zu allen Prozessoren im System sowie die gerechte Zuteilung bei Zugriffsanfragen für den Systembus sind die wichtigsten Gesichtspunkte eines Multiprozessorsystems. Eine Anzahl verschiedener Techniken wurde zur Implementierung dieses Systems bereits benutzt, wobei jede Lösung unter dem Gesichtspunkt der Hardware-Komplexität sowie des Kosten-Leistungsverhältnisses gesehen werden muß. Die Techniken unterscheiden sich im Decodierer/Arbiter:

- Anordnung: zentral oder lokal (bei jedem Prozessor);
- Typ: Anfrage-/Bewilligungslogik, Daisy-Chain-Logik, Parallel-Komparator.
- Anfrage-Identifikation: Anfrage-/Bewilligungsleitungen, räumliche Lage, Code.

Die Techniken mit der größten Leistung sind, wie üblich, diejenigen mit der höchsten Komplexität und den höchsten Kosten. Sie fordern außerdem einen umfangreichen Software-Support. Die Leistungsparameter eines Decodierers/Arbiters sind folgende:

- die Wartezeit: Periode zwischen Anfrage und Beginn der Ausführung;
- die Ausführungszeit: die erforderliche Zeit zur Ausführung der Anfrage, der Zuteilung und der Übertragung der Steuerung.

Die folgenden Beschreibungen zeigen, wie verschiedene existierende Mikroprozessorbusse die Funktionen der Prioritätscodierung und Abitration implementieren.

6.1 *Der Daisy-Chain-Decodierer/Arbiter des ZBI-Busses*

Der ZBI-Bus von Zilog ist eine 96polige Backplane-Verbindung, mit der Einzel- und Multiprozessorsysteme unterstützt werden [6]. Neben den 32 im Multiplexbetrieb benutzten Adreß/Daten- sowie 15 Stromversorgungsleitungen besitzt der Bus insgesamt 41 Steuerleitungen, mit denen folgende Funktionen unterstützt werden:

- konventionelle Datentransfers: AS*, DS*, R/$\overline{W}$, W/$\overline{LW}$;
- Interrupt auf drei Ebenen: INT1*, INT2*, INT3*, IE1, IE2, IE3, IO1, IO2, IO3;
- Systemsteuerung: MCLK, BCLK, N/$\overline{S}$, ST0, ST1, ST2, ST3, STOP*, RESET*;
- direkter Speicherzugriff: BAI*, BAO*, BUSREQ*;
- Aufteilung der Resourcen: MMAI*, MMAO*, MMREQ*, MMST*;
- Multiprozessor-Steuerung: CAI*, CAO*, CPUREQ*, CAVAIL;
- Fehlerprüfung: P0, P1, P2, P3, PE*, PWRBAD*.

Die Steuerleitungen für den direkten Speicherzugriff, die Aufteilung der Ressourcen sowie die Multiprozessorsteuerung ermöglichen den Aufbau eines dezentralisierten Priorität-Decodierers/Arbiters mit Daisy-Chain-Konzept für unterschiedliche Systemtypen. Die Steuerleitungen für den direkten Speicherzugriff sind vornehmlich für Systeme mit einem Master und mehrere Peripherieeinheiten in einer Prioritätskette gedacht. Die Steuersignale für die Aufteilung der Ressourcen und Multiprozessor-Steuerung sind beide für Mehrprozessor-Konfigurationen. Die Steuerleitungen für die Verteilung der Ressourcen sorgen dafür, daß immer nur ein Zugriff auf eine Ressource möglich ist. Die Multiprozessorsteuerung unterstützt ein aufwendigeres Protokoll von gegenseitigem Ausschluß und Transfer der Steuerung.

Die Steuerleitungen für die Aufteilung der Resourcen (MMREQ*, MMAI*, MMAO*, MMST*) ermöglichen den Aufbau von Multiprozessorkonfigurationen wie sie in Bild 6a dargestellt sind [7]. In dieser Konfiguration sind alle Prozessoren des Systems mit den Leitungen MMAI* und MMAO* verkettet. Low-Pegel auf der Leitung MMAI* (als Antwort auf einen MMREQ*) pflanzt durch die Kette fort, bis er den anfragenden Prozessor erreicht. Die Kombination von Low-Pegel auf MMREQ* und einem High-Pegel auf MMAI* identifiziert den neuen Bus-Master. Bevor ein Prozessor einen MMREQ* ausgeben kann, muß er die MMREQ*-Leitung abfragen. Wenn diese Leitung auf Low-Pegel liegt (busy), wird die Anfrage später wiederholt, oder sie entfällt. Wenn die Leitung auf High-Pegel liegt (frei) wird MMREQ* auf Low geschaltet. Die Anfrage war erst erfolgreich, wenn MMAI* nach einer vorherbestimmten Verzögerungszeit auch auf Low-Pegel geht. Anderenfalls wird die Anfrage abgebrochen.

Der eigentliche Decodierer/Arbiter besteht aus einer Kombination der in Bild 6b gezeigten Logikschaltung, der Z8000-Schaltung mit den Signalen µI* und µO*

sowie der Software mit den vier Befehlen MREQ, MRES, MSET und MBIT. Die vier Gatter und der Inverter bilden nicht nur die Prioritätssignale nach, sie formen auch die vier ZBI-Bus-Signale zu den zwei Multiprozessorsignalen des Z8000 µI* µO* um.

6.2 Der S-100-Parallel-Prioritätsdecodierer/Arbiter

Der S-100-Bus ist ein 100poliges Rückwandverbindungssystem, das für Einzel- und Multiprozessorkonfigurationen geeignet ist. Die 100 Busleitungen unterteilen sich in 16 Datenleitungen, 16 oder 24 Adressenleitungen, 10 Stromzuführungen, der Rest sind Steuerleitungen.

Der neue S-100-Bus, wie er vom IEEE Computer Society Standard Commitee (IEEE Tasks P692.1) definiert worden ist, enthält eine parallele Prioritäts-Decodierung/Arbitration mit hoher Geschwindigkeit für Multiprozessorsysteme mit bis zu 16 Mastern [8]. Der Bus bietet vier Prioritätsleitungen (DMA0* bis DMA3*), eine HOLD*-Leitung sowie ein pHOLDA (hold acknowledge). Jeder der Master muß einen Decodierer/Arbiter (*Bild 7*) sowie einen eigenen Prioritätscode besitzen. Die Arbeitsweise der Schaltungen und das dazu gehörige Protokoll, das in [8] beschrieben ist, können folgendermaßen zusammengefaßt werden:

- HOLD* darf nur benutzt werden, wenn es noch nicht signalisiert ist und pHLDA auf Low-Potential liegt;
- HOLD* muß entfallen, wenn pHLDA auf High schaltet und wenn ein anderer Controller höhere Priorität bekommt;
- HOLD* muß entfallen, wenn der Controller den Bus nicht länger benötigt;
- Die Priorität muß zugewiesen werden, wenn HOLD* auftritt, und muß bis zur nächsten fallenden Flanke von pHLDA zugewiesen bleiben;
- Die Prioritätsebene muß für den Benutzer auszuwählen sein (über Schalter) und auf den Busleitungen über Open-Kollektor-Treiber anliegen (DMA0*...DMA3*);

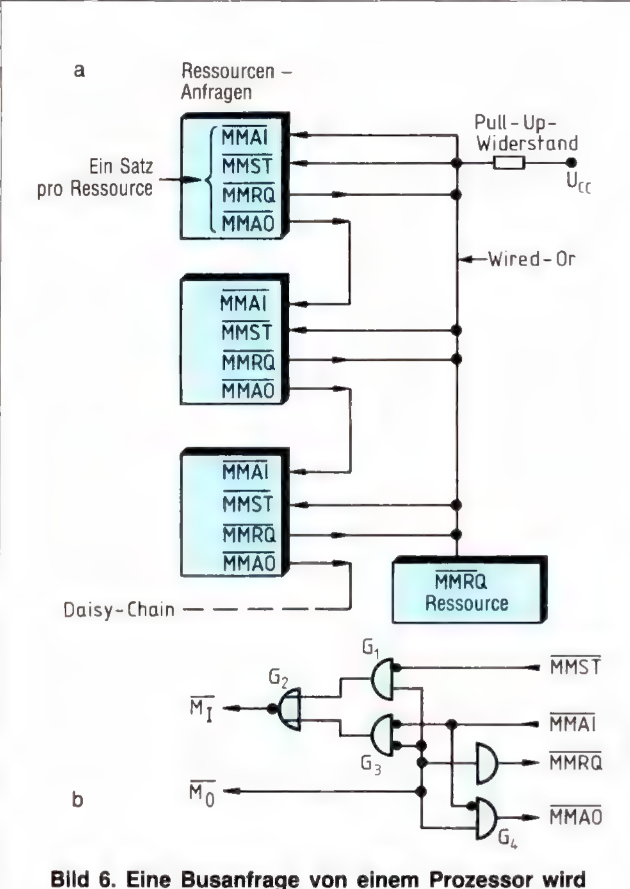


Bild 6. Eine Busanfrage von einem Prozessor wird über eine ODER-Verknüpfung auf die BUSRQ-Leitung gegeben und dazu benutzt, den Daisy-Chain-Prioritäts-Decodierer zu initialisieren. Der Prozessor, bei dem das Signal MMRQ* Low und das MMAO*-Signal High ist, bekommt die Steuerung des Busses (a), eine einfache Logik setzt die vier Bus-Steuerleitungen in die Daisy-Chain-Signale Mi* und Mo* des Z8000 um (b)

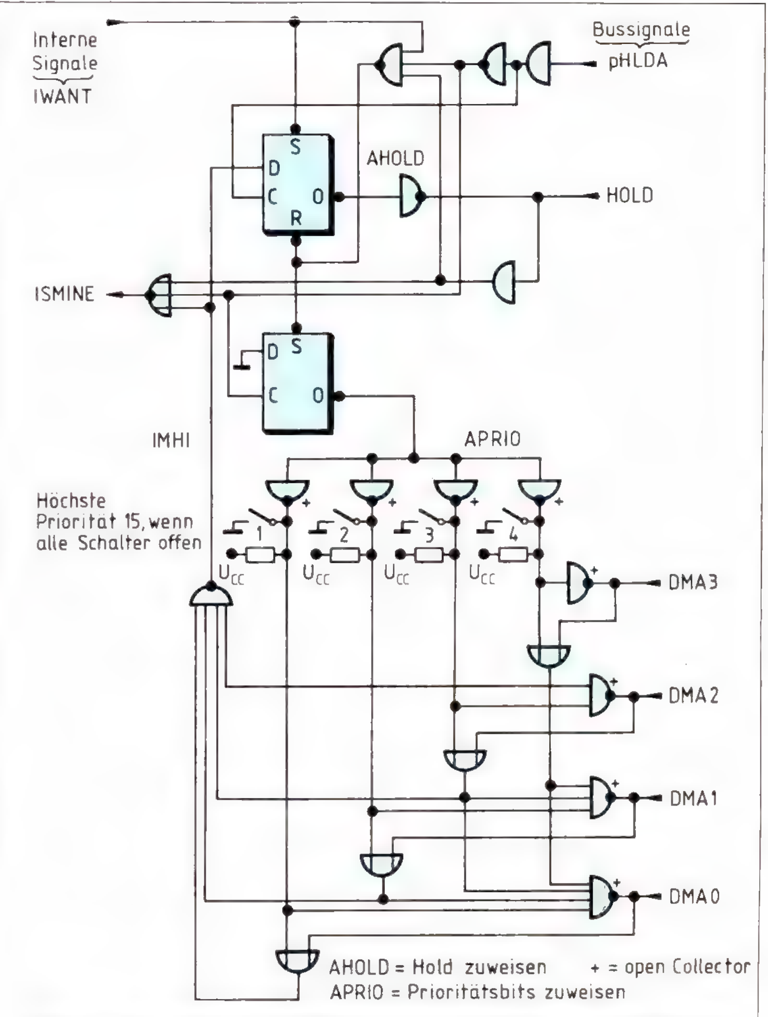


Bild 7. Der parallele Prioritätsdecodierer/Arbiter des S-100-Busses setzt die Prioritätsebenen mit vier Schaltern, die auf die Signale von DMA0*...DMA3* wirken und vergleicht seine eigene Prioritätsebene mit der auf dem Bus

- Das höchstwertige Bit der Prioritätsebene (erscheint auf DMA3*) muß mit der angegebenen Priorität verglichen werden. Falls die Leitung auf Low liegt, aber nicht durch den derzeitigen Master, müssen alle niedrigwertigen Prioritätsbits entfernt werden. Ähnlich müssen die Bits DMA2*, DMA1* und DMA0* überprüft werden und mögliche niedrigwertige Konflikt-Bits entfernt werden;
- Wenn keine der Leitungen auf Low gelegt sind, außer denen, die von dem derzeitigen Master auf Low geschaltet sind, hat dieser Master nach einer entsprechenden Einschwingzeit die höchste Priorität und kann den Bus übernehmen, wenn pHLDA auf High umschaltet
- Die Implementation der Logik muß so ausgeführt sein, daß die Einschwingzeit der Arbitrationsschaltung sowie des Busses zwischen dem HOLD*-Signal und dem Ansteigen von pHLDA liegt.

7 Software für Multiprozessorsysteme

Derzeit sind es erst relativ wenige Multi-Mikroprozessorsysteme in Betrieb. Ein Grund dafür ist, daß die Hardware, die am besten dafür geeignet ist, und zwar Einchip-Computer, die speziell für Mehrprozessorbetrieb konzipiert sind, zur Zeit noch nicht auf dem Markt erhältlich sind. Der andere Grund ist, daß die Systemarchitekturen noch entwickelt werden. Der Hauptgrund ist allerdings, daß nur sehr wenig Multiprozessor-Support sowie Anwender-Software erhältlich sind. Das Vorhandensein von zuverlässiger Unterstützungs-Software ist aber eine Vorbedingung für jede Entwicklungsarbeit. Das Aufkommen von strukturierten höheren Programmiersprachen (PASCAL, ADA usw.) sowie der modularen Betriebssysteme (RMX/86 usw.) für parallele und Multi-Tasking-Programmierung, verspricht, daß Lösungen nicht mehr lange auf sich warten lassen [9, 10, 11, 12].

7.1 Höhere Programmiersprachen

Reduzierung des Zeit- und Kostenaufwandes bei der Entwicklung von Anwenderprogrammen ist nur eines der Vorteile von höheren Programmiersprachen. Wichtiger ist die Aufteilung des Problems in immer kleinere Module nach der „Top-down-Methode" sowie die Prozeduren zur Kommunikation zwischen den Modulen. Diese Strukturen und Prozeduren sind die Basis für die Entwicklung jeglicher Soft- und Hardware. Diese werden insbesondere wichtig, wenn Betriebssysteme für Multi-Tasking- und Multi-Processing-Konzepte entwickelt werden.

7.2 Multiprozessor-Betriebssysteme

Die Funktion eines Betriebssystems ist die Verwaltung, Zuweisung und Vergabe der Ressourcen wie CPU-Zeit, Speicher sowie E/A-Einheiten. Ein Trend bei Betriebssystemen führt zu verstärkter Modularisierung. Sie bestehen aus einem Kern und einer Gruppe von Modulen, jeweils eines für jede Ressource im System. Betriebssysteme sind normalerweise als Software ausgeführt (auf Band oder Platte) oder als Firmware auf einem ROM. Moderne Entwicklungen bei Mikroprozessoren implementieren den Betriebssystemkern innerhalb der Hardware, direkt auf den CPU-Chip [13]. Das wird bei zukünftigen Multiprozessor-Entwicklungen Vorteile bringen, denn:

- es erfordert strengere Kommunikationsprozeduren zwischen den Modulen;
- es ermöglicht fertige Softwarelösungen und damit geringere Belastung des Softwareentwicklers;
- es räumt einen Teil der Hindernisse für die Akzeptierung von Multiprozessorsystemen aus dem Weg.

Obwohl Betriebssysteme für Multiprozessorprogramme solchen für Multi-Tasking-Programme sehr ähnlich sind, sind sie etwas komplizierter. Ein Punkt dabei ist, daß die gesamte Kommunikation zwischen den Modulen die Soft- und Hardwareverknüpfungen berücksichtigen muß. Ein weiterer Faktor zur Komplizierung ist die Koordination zwischen den Prozessoren, insbesondere in der Startphase, im leistungssparenden Betrieb (Power down) oder während oder bzw. nach Fehlern. Außerdem erfordern die meisten Anwendungen eine Art Synchronisation zwischen den Prozessoren. Weitere Schwierigkeiten treten bei dynamisch rekonfigurierbaren und fehlertoleraten Systemen auf. Eine Voraussetzung für solche Systeme ist ein Selbsttest und die Fähigkeit zur Fehlerisolierung. Ein Betriebssystem für ein Multiprozessorkonzept muß alle diese Funktionen enthalten.

Literatur

[1] *Satyanarayanan, M.:* Multiprocessing: An Annotated Bibliography. IEEE Computer Magazine, Mai 1980, S. 101...106.

[2] *Anderson, G. A.:* Computer Interconnection Structures: Taxonomy Characteristics and Examples. Computing Surveys, Bd. 7, Nr. 4, Dez. 1975.

[3] *Avizienis, A. et al:* Fault-Tolerant Digital System Design. Special Issue, Proceeding of the IEEE, Okt. 1978, S. 1107...1268.

[4] *Garrow, R.:* 16-Bit Single Board Computer Maintrains 8-Bit Family Tries. Electronics, 12. Okt. 1979, S. 105...110.

[5] MC68000 Family, Motorola Brochure M68KFM. 1980, Motorola Semiconductor Products Department, Austin, Texas.

[6] ZBI Z-Bus Backplane Interconnect System. Zilog Product Description, Juli 1980, Zilog Inc. Santa Clara, CA 95051.

[7] *Banning, J.:* Z-Bus and Peripheral Support Packages Tie Distributed Computer System Together. Electronic Design, 22. Nov. 1979, S. 144...149.

[8] *Elmquist, K. A. et al:* Standard Specification for S-100 Bus. Interface Devices, IEEE Computer Magazine, Bd. 12, Nr. 7, 1979.

[9] *Hemmenway, J.:* Advanced Software System Design Course, EDN, 20. Okt. 1979, S. 295...336.

[10] *Dorfner, F.:* PASCAL. ELEKTRONIK 1980, H. 3, S. 43...46.

[11] *Kuhn, K.:* Multitasking- und Multiprogramming-Betriebssystem für Mikrocomputer. ELEKTRONIK 1980, H. 2, S. 69...73.

[12] *Mauthe, R.:* Betriebssystem unterstützt modernen 16-Bit-Computer (iRMX86). ELEKTRONIK 1981, H. 3, S. 73...80.

[13] *Geyer, J.:* 32-Bit-Mikrocomputer besitzt neuartige Architektur (iAPX432). ELEKTRONIK 1981, H. 5, S. 59...66.

Hermann Schmid

Multi-Mikroprozessor-Systeme

2. Teil: Technischer Stand

Der 1. Teil dieser Aufsatzreihe beschäftigte sich mit den Grundlagen, die zunächst kurz zusammengefaßt dargestellt werden sollen: MIMD-Architekturen (Multiple Instruction, Multiple Data) werden in vier Gruppen aufgeteilt: a) Mehrere CPUs teilen sich einen gemeinsamen Speicher, b) mehrere Prozessoren arbeiten auf einem gemeinsamen Bus, c) hierarchische Systeme, d) Kreuzschienen-Netzwerke. Jede Gruppe kann mit verschiedenen Bus- und Speichertypen sowie Konfigurationen und Operationen implementiert werden, um verschiedene Ebenen der Verkopplung bzw. Leistungsstufen zu erreichen. Die Systemeffizienz ist sehr stark abhängig von den Techniken, die für Prioritätscodierung und Busarbitration, Datentransfer sowie Steuerung der einzelnen Prozessoren verwendet werden. Außerdem ist es die Software – die Programmiersprachen sowie die Möglichkeiten des Betriebssystems –, die die gesamte Architektur, die Implementationen, Operationen und Merkmale bestimmt.

1 Am weitesten verbreitet: Gemeinsamer Bus im Time-Share-Betrieb

Von den verschiedenen Klassen und Kategorien der Multi-Mikroprozessor-Architekturen, die bereits aufgezählt wurden, findet man die Gruppe 2b vom Typ MMPS-1 [10] heute am häufigsten. Diese Gruppe, die im allgemeinen als Architektur mit gemeinsamen Bus im Time-Share-Betrieb (TSCB – *Time Shared Common Bus*) bezeichnet wird, kann entweder mit einem parallelen oder seriellen Bus implementiert werden, außerdem mit oder ohne gemeinsamen Globalspeicher. Derzeit übliche TSCB-Systemimplementationen verwenden in den meisten Fällen einen Parallelbus (*Bild 1*).

In existierenden TSCB-Systemen werden N unabhängige Mikrocomputer entweder durch einen parallelen (Multibus, Versabus usw.) oder seriellen Bus verbunden. Jeder dieser Mikrocomputer (8 oder 16 Bit) ist eine physikalische Einheit – eine Platine oder ein Chip –, besitzt eine CPU mit entsprechender Steuerung, einen lokalen Bus, lokalen oder globalen Speicher, serielle sowie parallele E/A-Einheiten, Zähler/Zeitgeber, ein Systembus-Interface mit Prioritätscodierungs- und Busarbitrationsschaltungen. Wie viele Prozessoren können auf diese Weise verbunden werden? Die Zahl N hängt von vielen Parametern ab, wie das später beschrieben werden wird, sie ist aber im allgemeinen kleiner als acht.

Der Systembus ist eine allgemeine Ressource, die von allen angeschlossenen Einheiten benutzt wird. Nur ein Computer kann ihn zu einer Zeit benutzen. Die Bussteuerung muß abgefragt, zugeteilt und vom derzeitigen

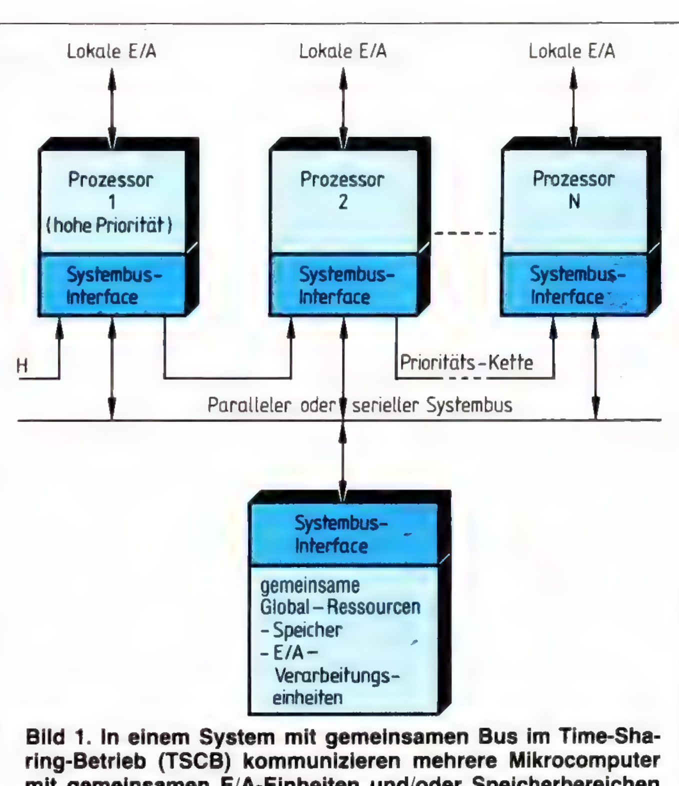


Bild 1. In einem System mit gemeinsamen Bus im Time-Sharing-Betrieb (TSCB) kommunizieren mehrere Mikrocomputer mit gemeinsamen E/A-Einheiten und/oder Speicherbereichen miteinander über den Systembus

zum neuen Master übergeben werden. Die verschiedenen Systeme, die hier beschrieben werden, unterscheiden sich insbesondere in der Weise, wie die Computer miteinander kommunizieren und wie die Bussteuerung zwischen ihnen übergeben wird. In Systemen mit hoher Effizienz muß der Busarbitration sehr schnell, die Auswahl des Partners in einem Vorgang und der Datentransfer in möglichst kurzer Zeit durchgeführt werden. Ein wichtiger Gesichtspunkt bei der Aufteilung des Anwenderprogramms ist die Minimierung der Zahl globaler Variablen, die zwischen den Computern transferiert werden müssen. Prioritätscodierung, Arbitration und Bus-Steuerfunktionen sind verteilt, wenn keine zentrale Steuereinheit vorgesehen ist. Alle Steuerfunktionen werden von den am Bus angeschlossenen Einheiten ausgeführt.

Die Systemeffizienz hängt auch sehr stark von der Art des Speichers ab, der für die globalen Variablen benutzt wird. Der Globalspeicher kann zentral (für alle) oder verteilt bei den N Prozessoren angeordnet sein. Im letzten Fall beeinflußt die Aufteilung und die Art des Zugriffes durch andere Prozessoren die Gesamteffizienz. Eindeutige Adressierung ist eine Grundvoraussetzung, die durch Mapping des Speichers eines jeden Prozessors in den globalen Speicherbereich erreicht wird (siehe Teil 1, Bild 4). Auf diese Weise kann der Speicher eines jeden Partners durch Hinzufügen von wenigen Bits zur lokalen Adresse adressiert werden. Um lokale Daten und Programme zu schützen, ist es wünschenswert, den Speicherbereich, der globale Variable enthält, vom Rest zu trennen und ausschließlich darauf den Zugriff zu beschränken. Die erforderliche Zeit für den Zugriff auf den Speicher eines Partners wird sehr stark reduziert, wenn die globalen Variablen in einem Dual-Port-Speicher untergebracht sind.

2 TSCB-System mit Einplatinen-Computern

Für jeden Mikroprozessortyp gibt es verschiedene Einplatinen-Computer. Allerdings haben nur wenige dieser Boards Einrichtungen zum Multiprozessorbetrieb. Wenn die Anwendung die Möglichkeiten eines einzelnen Computers überschreiten, können zwei oder mehrere miteinander verbunden werden und als ein Multiprozessorsystem arbeiten. Dies gilt insbesondere für die neuen 16-Bit-Mikroprozessoren (8086, Z8000, 68000) weil diese CPUs besondere Hard- und Softwaremerkmale für Multiprozessorbetrieb besitzen. Beispielsweise bietet der Typ Z8000 vier Signalleitungen (BUSRQ*, BUSAK*, MI*, MO*) sowie vier Instruktionen (MBIT, MERQ, MRES, MSET), um Bus-Anfragen, Prioritätscodierung sowie Busarbitrations-Operationen auszuführen.

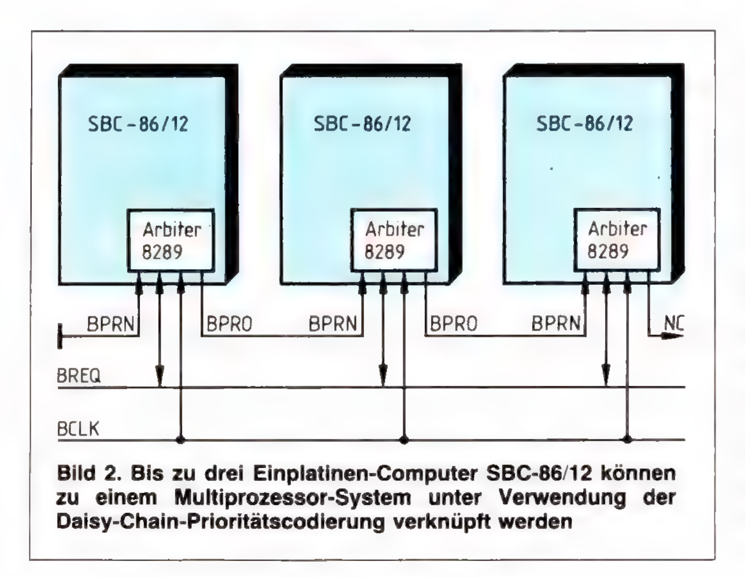


Bild 2. Bis zu drei Einplatinen-Computer SBC-86/12 können zu einem Multiprozessor-System unter Verwendung der Daisy-Chain-Prioritätscodierung verknüpft werden

2.1 Multiprozessorbetrieb mit den Single-Board-Computern von Intel

Die meisten Einplatinencomputer von Intel sind Multibus-kompatibel. Nur wenige besitzen die Möglichkeit zum Multiprozessorbetrieb. Der Typ SBC 86/12, der eine Dreifach-Bus-Architektur besitzt, basiert auf der 16-Bit-CPU 8086, bietet 32 KByte „Pseudo-Dual-Port-Memory“, serielle sowie parallele E/A-Einheiten und das Multimaster-Businterface. Der Dual-Port-Controller verbindet entweder den Systembus oder den lokalen Bus mit dem Speicher. Wenn dieser am Multibus angeschlossen ist, werden externe Adressen durch Hinzufügen von Displacements ergänzt. Deswegen kann nur eines von vier Segmenten vom Multibus adressiert werden [11].

Der SBC 86/12 verfügt über einfache Busarbitrationen, die serielle Prioritätscodierung (*Daisy-Chain*) auf der Platine benutzt (*Bild 2*). Die Anzahl der Prozessoren (N), die auf diese Weise kombiniert werden können, wird durch die Verzögerungszeit des höchsten Prioritätsausgangs (BPRO*) zum Eingang mit geringster Priorität (BPRN*) auf drei begrenzt. Diese Verzögerung kann 100 ns oder eine Bus-Taktperiode von 10 MHz nicht überschreiten. Tatsächlich können bei geringerer Taktfrequenz des Busses mehr Prozessoren aneinander geschaltet werden, allerdings geht die Systemeffizienz sehr stark zurück [12].

Eine wesentlich bessere Möglichkeit, um die Anzahl der Prozessoren in einem System zu erhöhen, ist die Verwendung einer parallelen Bus-Arbitrationslogik (extern am SBC 86/12), wie sie für den S-100-Bus (siehe Teil 1, Bild 7) definiert ist oder vom Busarbiter-Baustein 8289 von Intel (*Bild 3*) geboten wird. Hier werden bis zu 16 Bus-Request-Leitungen mit einem zentralen Prioritätscodierungs-Netzwerk verbunden (bestehend aus 2 × 74148 und 2 × 74138). An dessen Ausgängen liegen die BPRN*-Signale an. Mit jeder dieser Methoden ist es dem derzeitigen Bus-Master immer möglich, seine Aktivitäten auf dem Bus zu beenden, bevor er die Steuerung einem Master mit höherer Priorität übergibt. Außerdem kann der Bus-Master die Kontrolle über den Bus durch „Verriegelung“ behalten. Diese Möglichkeit wird benötigt, wenn kritische Funktionen beim Testen oder Setzen von Semaphoren oder Ausführen bestimmter E/A-Funktionen vorliegen.

2.2 Multiprozessorbetrieb mit dem SBC-8000

Der Einplatinen-Computer SBC-8000, der vom Verfasser konstruiert wurde, bevor solche Platinen von Zilog, AMD oder anderen Firmen verfügbar waren, besitzt eine CPU Z8002, 16 KByte Programmspeicher und 4 KByte Datenspeicher, fünf 16-Bit-Zähler, eine serielle RS-232-Schnittstelle sowie 32 diskrete E/A-Anschlüsse. Die Architektur benutzt eine dreifache Busstruktur: den ungepufferten Z8000-Bus, den lokalen Z-Bus und den ZBI-Bus (*Zilog Backplane Interconnect*).

Zwei Gruppen von Puffern verbinden wie ein mehrpoliger Umschalter den ungepufferten Z8000-Bus mit dem lokalen Bus, wenn ein Memory-Request-Signal (MEMRQ) vorliegt, oder mit dem Systembus, wenn ein E/A-Request-Signal (IORQ) vorliegt. Speicher und E/A-Funktionen sind lokal angeordnet, und die Memory-Mapped-Ressourcen befinden sich am lokalen Bus. Alles, was an den Systembus angeschlossen ist, sind E/A-Mapped-Global-Ressourcen, auf die nur mit einer E/A-Operation zugegriffen werden kann [13].

Um die Hardware des SBC-8000 zu beschränken, war ein wichtiger Gesichtspunkt bei der Entwicklung die Ausnutzung der oben erwähnten Merkmale in der Hard- und Software des Z8000, die den Multiprozessorbetrieb unterstützen: Daisy-Chain-Prioritätscodierung sowie firmwareimplementierte Arbitration (*Bild* 6). Um einen SBC-8000 mit dem Bus verbinden, muß das Programm eine MEMRQ-Instruktion ausführen, die die erste MI*-Leitung (*Micro In*) testet, darauf die MO*-Leitung (*Micro Out*) zuweist und nach einer vorher festgelegten Verzögerungszeit die MI*-Leitung erneut testet. Wenn beim zweiten Test das MI*-Signal auf High liegt, wird der Anfrager zum Bus-Master [14].

Wenn der Zugriff auf den Bus erfolgt ist, wählt der Master seinen Partner aus, indem er eine 3-Bit-Adresse auf den Systembus legt (*Bild* 4). Die Adreß-Decodierungslogik des ausgewählten Prozessors erzeugt ein Bus-Request-Signal, das dafür sorgt, daß die CPU die Kontrolle über ihren lokalen Bus nach nicht mehr als 10 Taktzyklen (2,5 μs) wiedererlangt. Sobald der Partner dieses mit dem BUSAK-Signal (*Bus Acknowledge*) quittiert, kann der Master den Austausch von Daten mit den Ressourcen des Partners starten, wobei normale E/A-Operationen benutzt werden.

Firmwareimplementierte Arbitration und Zugriff auf den Speicher des Partners mit Bus-Request ist langsam. Die Ausführung der MEMRQ-Instruktion kann eine Zeit von bis zu 10 μs beanspruchen, weil eine interne Verzögerung von mindestens 3 μs (2,5 μs ist die maximale Reaktionszeit für BUSRQ) in der MEMRQ-Instruktion vorgesehen werden muß. Daraus ergibt sich eine erfolgreiche Gesamtzeit, die für den Zugriff auf ein einzelnes Wort aus dem Speicher des Partners benötigt wird, von typisch 20 μs, was viel zu lang ist für einen effizienten Datentransfer.

Die Zugriffszeit zum Bus und zum Speicher des Partners kann um einen Faktor von 2 reduziert werden, wenn parallele Prioritätscodierer/Arbiter sowie ein echtes Dual-Port-RAM benutzt werden. Diese Verbesserung rechtfertigt den relativ kleinen Aufwand an Hardware und für die zusätzlichen Bus-Prioritäts-Leitungen (drei für ein 8-Prozessorsystem). Allerdings sind echte Dual-Port-RAMs derzeit noch nicht verfügbar. Ein solches Bauelement, ein 4-K×8-RAM, wird von Motorola im Laufe des Jahres 1982 eingeführt. Wenn nur ein paralleler Prioritätscodierer/Arbiter verwendet wird, kann die Zeit zum Datentransfer bereits wesentlich verringert werden (*Bild* 5).

3 TSCB-Systeme mit Einchip-Controllern

Einchip-Controller (SCC – *Single Chip Controller*) wie z. B. Z8, 6801, 6500/1, 8051 oder 9940 sind komplette Prozessoren mit CPU, Daten-/Programmspeicher, serieller und paralleler E/A-Einheit und für Anwendungen mit großen Stückzahlen und geringem Verkaufspreis konzipiert (Werkzeuge, Spiele, Automobile usw.). Die Leistung ist begrenzt, aber trotzdem recht beeindrukkend: Ein 8051 führt beispielsweise eine 8-Bit-Registeraddition in einer μs und einem 8-Bit-Multiplikation in vier μs aus. SCCs bieten das geringste Kosten/Leistungsverhältnis [15].

Es ist nicht überraschend, daß häufig versucht wird, eine Anzahl von SCCs in Multiprozessorsystemen zu verknüpfen, die nicht nur höhere Leistung als ein einzelner SCC bieten, sondern schrittweise erweiterungsfähige Merkmale (Durchsatz, Speicher- und E/A-Kapazität) aufweisen, um Anforderungen unterschiedlichster Anwendungen in möglichst kosteneffektiver Weise entsprechen zu können.

Unglücklicherweise sind die meisten SCCs nicht für Multiprozessor-Betrieb konzipiert. Nur die 8070-Typen

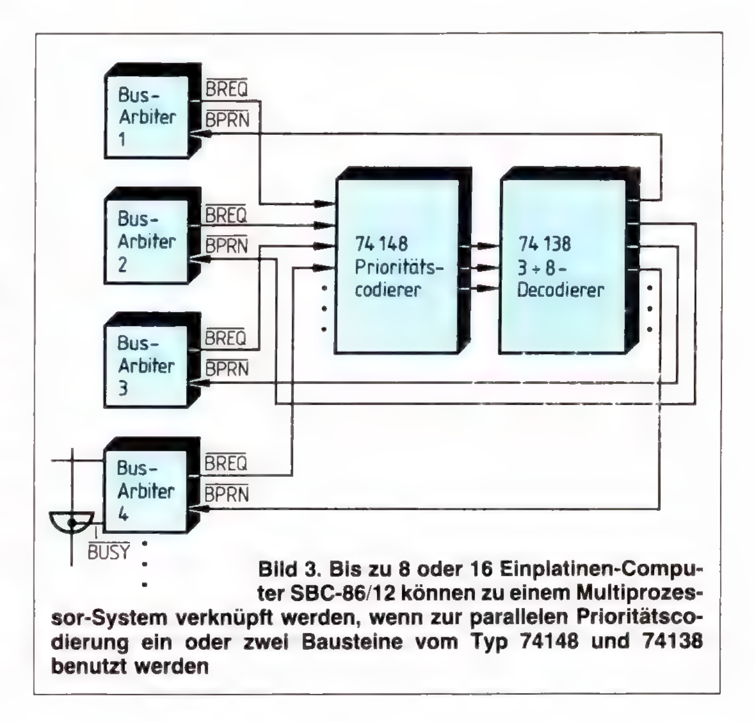


Bild 3. Bis zu 8 oder 16 Einplatinen-Computer SBC-86/12 können zu einem Multiprozessor-System verknüpft werden, wenn zur parallelen Prioritätscodierung ein oder zwei Bausteine vom Typ 74148 und 74138 benutzt werden

sowie die 8051-Familie haben begrenzte Möglichkeiten für den Multiprozessor-Betrieb. Aus diesem Grunde existieren nicht nur alle oben erwähnten Probleme, die für die Einplatinen-Computer gelten, sie werden sogar noch verstärkt. Die Kommunikation zwischen den Prozessoren ist beispielsweise besonders langsam bei Architekturen, die einen seriellen Bus benutzen. Trotzdem bieten serielle Bussysteme eine akzeptable Lösung bei Applikationen mit umfangreichen, unabhängigen Tasks und wenigen globalen Variablen, die ausgetauscht werden müssen.

Die Typen Z8, 6801 sowie 8051 sind 8-Bit-Controller mit 128 Byte RAM und 2048 Byte ROM, einem oder zwei Zeitgebern, Taktgenerator, UART sowie bis zu 32 programmierbaren E/A-Anschlüssen, die in vier 8-Bit-Ports aufgeteilt sind. Unterschiede bestehen in Taktfrequenz, Ausführungszeiten, RAM-, Register- und Portorganisation, Befehlssatz sowie der Prototypen-Ausführung. Intel und Motorola bieten ein EPROM auf dem Chip an, während Zilog einen Baustein mit aufsteckbarem EPROM entwickelt hat.

3.1 Multiprozessorbetrieb mit den Einchip-Computer NS 807X

Die 8070-Familie von National Semiconductor besteht aus verschiedenen Prozessoren, die alle eine 8-Bit-CPU, verschiedene 8/16-Bit-Register, 64 Byte RAM, Taktgenerator auf dem Chip, Bus-Zugriffslogik sowie unter-

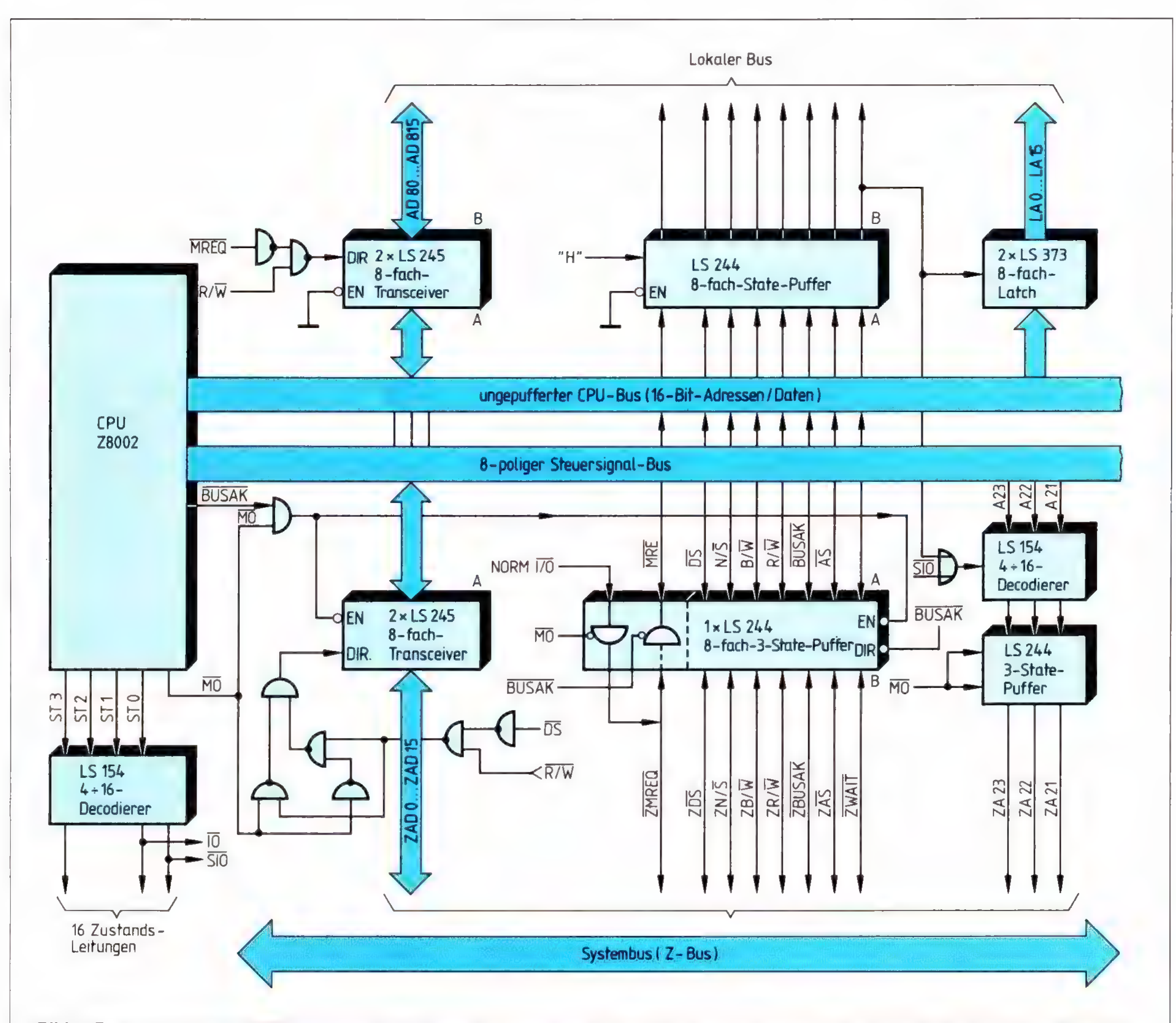


Bild 4. Der ungepufferte Z8000-Bus wird durch zwei Gruppen von Puffern, die wie ein mehrpoliger Umschalter wirken, entweder mit dem Z-Bus oder dem ZBI-Systembus (Zilog Backplane Interconnect) verbunden

schiedliche ROM-Kapazitäten bieten: der Typ 8070 hat kein, 8072 insgesamt 2,5 KByte, der 8074 eine Kapazität von 4 KByte ROM. Auf dem Chip befinden sich keine UART und kein Zähler. Diese Bausteine unterscheiden sich von anderen SCCs, die isoliert betrieben werden können, weil sie folgende Merkmale besitzen:

- Ein permanentes Businterface (16 Adressen, 8 Datenleitungen) anstelle der konventionellen programmierbaren E/A-Leitungen.
- Spezielle Wortverarbeitungs-Instruktionen wie z. B. „Suche und springe, falls das Zeichen übereinstimmt" oder „Verzweigung falls kein Zeichen vorliegt".
- Besondere Signale, z. B. Bus-Request, Freigabe von Ein- und Ausgang, um den Multiprozessorbetrieb zu unterstützen.
- Relativ schnelle 16-Bit-Multiplikation (37 µs) sowie -Division (42 µs).

Multiprozessorbetrieb mit den SCCs aus der 8070-Familie wird durch die Bus-Zugriffs-Logik, dem ILD- (Inkrement und Laden) sowie DLD-Befehl (Dekrement und Laden) außerordentlich stark vereinfacht. Unter Verwendung von Daisy-Chain-Codierung können verschiedene 8070-Bausteine über einen Parallelbus und die Bus-Request-Leitung verbunden werden. Es sind keine zusätzlichen Bauelemente notwendig [16].

Die einzelnen Bausteine vom Typ 8070 in einem Multiprozessorsystem (Bild 6) werden durch Anschluß des ENOUT*-Ausganges einer Einheit mit höherer Priorität an den ENIN*-Eingang des nächsten Bausteines in der Linie priorisiert. Wenn ein Prozessor den Bus benötigt, muß er das BERQ*-Signal überprüfen, bevor einer der Speicher-Referenz-Funktionen ausgeführt wird. Wenn BREQ* nicht aktiv ist, zieht der Prozessor 8070 diese Leitung auf Low-Potential und überprüft sein ENIN*-Signal. Im einzelnen kann folgendes eintreten:

- Wenn ENIN* auf High-Potential liegt, wird ENOUT* auf High gesetzt und dem 8070 der Zugriff auf den Bus gesperrt.
- Wenn ENIN* auf Low-Potential liegt, wird ENOUT* auf High gesetzt und dem 8070 wird der Zugriff auf den Bus gestattet.

Diese Prozedur arbeitet in Zusammenhang mit speziellen Softwareoperationen auch, wenn verschiedene Prozessoren gleichzeitig auf den Bus zuzugreifen versuchen. Wenn der Prozessor mit der höchsten Priorität seine Operationen beendet hat, bzw. einen integralen Teil davon (z. B. ein Byte in einer ILD-Instruktion aufnehmen), gibt er den Bus wieder frei, indem er die BREQ*-Leitung auf High-Potential schaltet, so daß der nächste Prozessor, der an der Reihe ist, auf den Bus zugreifen kann. Aus diesem Grund wird jeder wartende Prozessor berücksichtigt, bevor neue Bus-Requests erlaubt sind. Dieses Verfahren vermeidet das Problem des Ausschlusses und minimiert die Zeit, die ein Prozessor auf dem Buszugriff warten muß.

Ein gemeinsamer Speicher ermöglicht den Austausch globaler Variabler zwischen den Prozessoren. Auf den Speicher kann von allen Prozessoren aus über normale Schreib-Lese-Operationen zugegriffen werden. Die 16 Adreßleitungen erlauben direkte und schnelle Adressierung des internen ROMs (0000...09FF) sowie RAMs (FFC0...FFFF) und eines externen RAMs (0FFF...FFBF). Die ILD- und DLD-Instruktionen ermöglichen Read-Modify-Write-Operationen durch spezielle Flags (Speicherplätze), um Semaphoren-Behandlung zu unterstützen.

Derzeit ist die einzige Literatur, in der die Multiprozessor-Operationen dieser Familie beschrieben werden, das Datenblatt des INS8070 [17], allerdings ist ein ausführlicheres Applikations-Handbuch in Vorbereitung.

3.2 *Multiprozessor-Betrieb mit dem Einchip-Computer 8051*

Ein bisher einmaliges Merkmal für einen Baustein dieser Art ist das Kommunikationsprotokoll der UART im 8051 von Intel [18]. Damit ist es möglich, eine große Anzahl Prozessoren über einen seriellen Bus zu verbinden. Das dadurch entstehende System ist, obwohl für räumlich getrennte Prozessorsysteme konzipiert, besonders interessant, weil es echte blockweise Erweiterungsfähigkeit besitzt, ohne daß zusätzliche Bauelemente erforderlich sind.

Die Hardware der UART des 8051 ist konventionell aufgebaut. Die Multiprozessorfähigkeit wird durch mehrere Ebenen im Protokoll gegeben. Das Register für die Steuerung des seriellen Ports (SCON) erlaubt z. B.: 1.

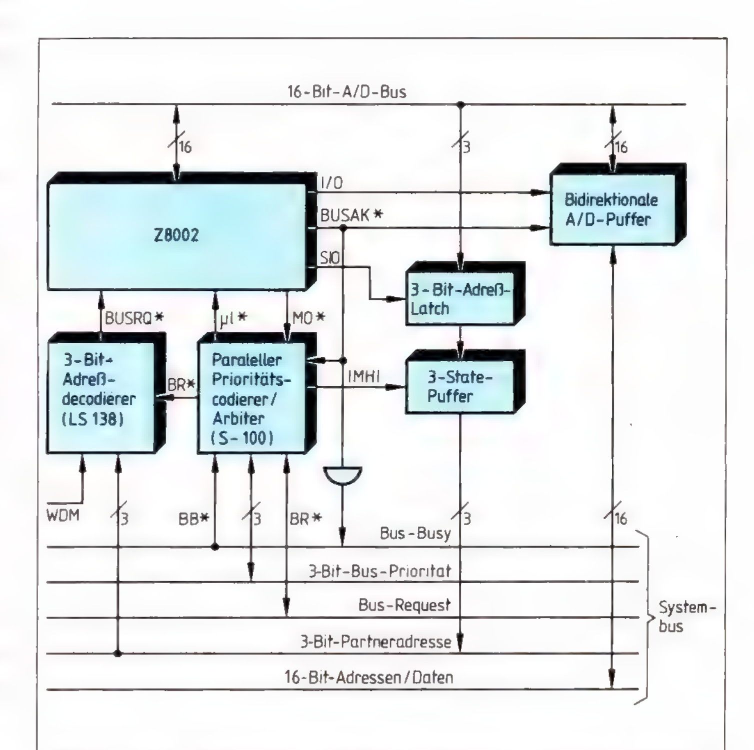


Bild 5. Ein S-100-Parallel-Prioritätscodierer und -Arbiter paßt in einfacher Weise die MI*- und MO*-Leitungen des Z8000 an und erfordert nur drei zusätzliche Prioritätsleitungen auf dem Systembus

Auswahl der vier Betriebsarten mit den SM0- und SM1-Flags, 2. Konditionierung und Freigabe des Empfängers mit den REN- und SM2-Flags, 3. Setzen des Empfänger-/Sender-Bit 8 mit RB8 und TB8 sowie 4. Steuerung der Flags RI und TI für Sende-/Empfangs-Ende. Die Betriebsart 3 (11-Bit-Übertragung mit konstanten 187,5 kBaud) wurde insbesondere für die Kommunikation zwischen den Prozessoren konzipiert. Wenn das neunte Rahmen-Bit gesetzt ist, aktiviert ein Reception-Complete-Interrupt (RI) den Prozessor.

Die Kommunikation des 8051 mit anderen Prozessoren [19] basiert auf einem Master-Slave-Protokoll, in dem ein Master und mehrere Slaves vorgesehen sind. Das Protokoll umfaßt folgende Schritte:

- Der Master überträgt eine 8-Bit-Adresse des Slave, die durch ein aktives Bit TB8 im SCON bezeichnet ist.
- Jede UART der Slaves wartet auf die Adresse. Wenn diese vorliegt und wenn das Bit SM2 gesetzt ist, unterbricht sie die CPU, die die Adresse mit der eigenen vergleicht.
- Der angewählte Slave setzt in seinem SCON das Bit SM2 zurück, um alle folgenden Aussendungen des Masters zu empfangen.
- Der Master sendet die Steuerinformationen wie „Schreiben“ oder „Lesen“ sowie die Daten.

Die Netzwerkeffizienz kann darüber hinaus noch verbessert werden, wenn Software definiert wird, die Steuerbits in der Adresse erkennt. Wenn z. B. Bit 7 des Adreßrahmens für den Master gesetzt ist, sind alle Slaves zum Empfang der folgenden Nachricht freigegeben, wenn Bit 7 logisch Null ist, kann nur der angewählte Slave die Nachricht empfangen. In ähnlicher Weise zeigt das Bit B7 im Slave-Rahmen das Erkennen eines Kommandos mit einer 1 oder einen Fehler mit einer 0.

Typisch für Multi-Mikroprozessorsysteme mit dem 8051 ist das Master-Slave-Prinzip. Dabei ist die gesamte Steuerung verteilt und befindet sich in den UARTs der Prozessoren und deren Software. Die Master-Slave-Architektur schließt Zugriffskonflikte auf dem Bus aus, weil der jeweilige Master die Steuereinheit darstellt und bestimmt, was jeder Slave tun muß und in welcher Reihenfolge dies zu geschehen hat.

Mit einer Rate von 187,5 kBaud und in einem Rahmen von 11 Bit können Datenbytes zwischen Master und Slave innerhalb von 58 μs übertragen werden. Um Daten zwischen zwei Slaves transferieren zu können, muß der Master erst ein Kommando zum aussendenden Slave senden, bevor die eigentliche Übertragung beginnen kann. Das hat zur Folge, daß ein Multi-Mikroprozessorsystem mit Einchip-Computern vom Typ 8051 nur langsamen Datentransfer über den seriellen Bus unterstützen kann.

Die UART des 8051 unterstützt sowohl Duplex- als auch Halb-Duplex-Datenübertragung (*Bild 7*). Theoretisch können in einem solchen System bis zu 128 Prozessoren kombiniert werden. Praktisch ist die Zahl durch die Treiberkapazität der Transmitter sowie die Kapazitäts- und Rauscheigenschaften des Verbindungskabels begrenzt. Die tatsächlich realisierbare Anzahl der kombinierbaren Einzelprozessoren hängt daher von der Kabellänge, der Abschirmung und dem elektromagnetischen Störpegel der Umgebung ab.

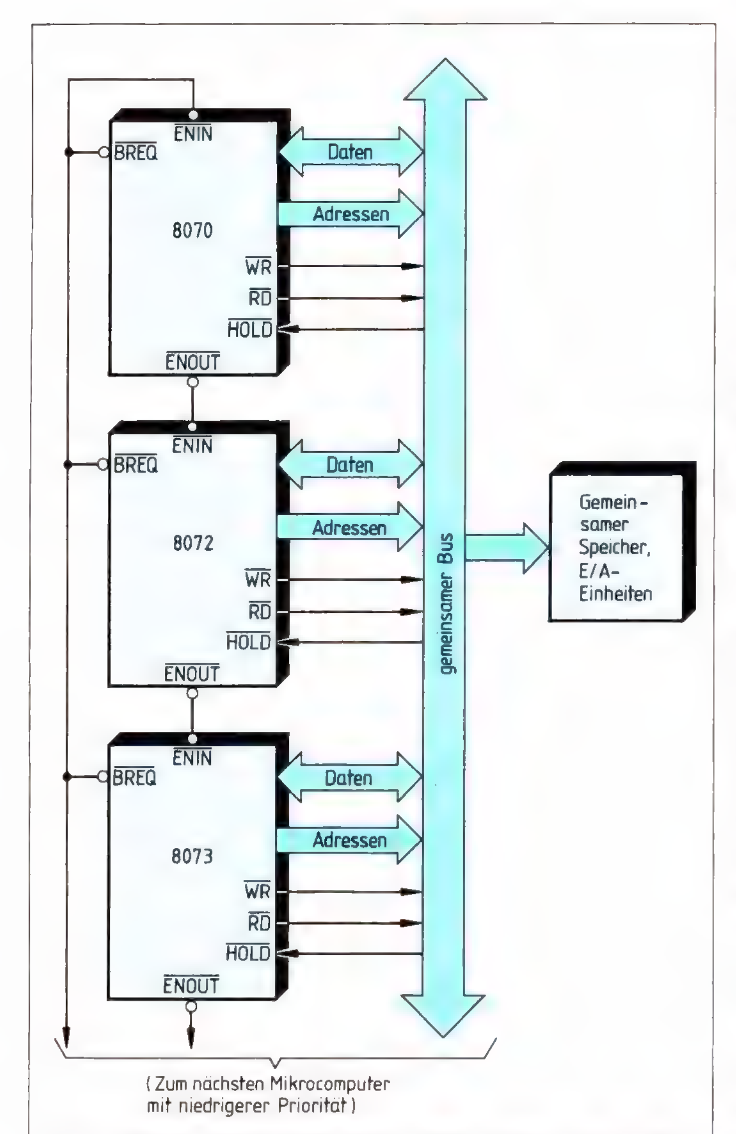


Bild 6. Mehrere Einchip-Controller 807X werden über einen 24poligen Adreß-/Datenbus sowie eine einzelne Bus-Request-Leitung miteinander verknüpft. Die Freigabeanschlüsse werden hintereinandergeschaltet (Daisy-Chain), so daß der oberste Prozessor die höchste und der unterste die niedrigste Priorität hat

4 Multi-Mikroprozessor-Software

Software für Systeme mit einem einzigen Prozessor besteht aus dem Anwendungsprogramm, dem Betriebssystem, das die Steuerung oder die Programme zur Verwaltung der Ressourcen (Supervisor, Executive usw.) sowie die Software-Entwicklungsunterstützung (Übersetzer, Dienst- und Diagnose-Programme) umfaßt (*Bild 8a*).

Multiprozessor-Software (Bild 8*b*) unterscheidet sich davon insbesondere im Betriebssystem. Multiprozessor-

Betriebssysteme (MPOS) müssen lokale und globale Ressourcen verwalten sowie die Kommunikation zwischen den Prozessoren steuern können. Das Maß der Verantwortung in der Verwaltung hängt sehr stark von der Architektur des Multi-Mikroprozessor-Systems, dem Grad der Kopplung zwischen den Prozessoren sowie davon ab, ob die Steuerung zentral oder verteilt angeordnet ist [20].

Die Steuerfunktionen eines Multiprozessor-Betriebssystems müssen dafür sorgen, daß die Tasks, die in den verschiedenen Prozessoren zu verschiedenen Zeiten ausgeführt werden, nicht miteinander in Konflikt geraten und das System zum Zusammenbruch oder Blockieren bringen. Im einzelnen muß ein solches Betriebssystem folgende Funktionen ausführen:

- Synchronisierung der Prozessoren und Prozesse durch zeitliche Planung der Tasks (Scheduling), Priorisierung und ihre Ausführung;
- Verwaltung und Aufteilung der lokalen sowie globalen Ressourcen (Speicher, CPUs, Busse, E/A-Einheiten usw.), abhängig vom Grad der Verkopplung der Prozessoren;
- Bereitstellung von Einrichtungen für Diagnose, Rekonfiguration und Wiederaufnahme des Betriebes, um bestimmen zu können, welches Element fehlerhaft arbeitet, dieses abzuschalten und das System neu zu starten.

4.1 *Weiterentwicklung von Multi-Tasking-Betriebssystemen (MTOS) zu Multiprozessor-Betriebssystemen (MPOS)*

Echtzeit-Multi-Tasking-Betriebssysteme gibt es mittlerweile für die meisten 8- und 16-Bit-Mikroprozessorfamilien (iRMX 80/86, Versados, ZRTS/Zeus usw.), die die üblichen Funktionen des Ressourcen-Scheduling, der Steuerung und Verwaltung übernehmen. Der einfachste Weg, ein MPOS zu produzieren, ist die Multiprozessor-Steuerfunktionen einem existierenden MTOS hinzuzufügen. Die Firma Industrial Program Inc. (IPI) macht genau dieses und vermarktet eine ganze Familie von Multiprozessor-Betriebssystemen [21].

MTOS-68k, das Echtzeit-Betriebssystem für die Prozessorfamilie 68000 von Motorola, ist das neueste dieser Produkte. Es bietet nicht nur die Intelligenz, die es ermöglicht, daß mehrere Prozessoren als ein System funktionieren, es führt dieses auf eine Weise aus, daß das Anwendungsprogramm nicht berücksichtigen muß, wieviele Prozessoren im System vorhanden sind. MTOS-68k ist für Systeme mit 1...16 Prozessoren konzipiert, die alle gleichwertige Partner sind. Es gibt keine Master oder Slaves. Das MTOS wird in jedem Prozessor gespeichert.

Das Betriebssystem behandelt jeden Prozessor als eine Ressource, der Tasks zugeordnet werden können. Die Task mit der höchsten Priorität wird immer dem ersten verfügbaren Prozessor zugewiesen. Eine typische Betriebssituation könnte folgendermaßen ausschauen: Task X läuft auf Prozessor P und wird unterbrochen, um eine höherpriorisierte Task Y auszuführen. Wenn Y abgeschlossen ist, wird die Ausführung von X weiterbearbeitet, aber möglicherweise auf einem anderen Prozessor. Im allgemeinen ist es nicht möglich, vorauszusagen, welcher Prozessor gerade welche Task ausführt, wenn nicht besondere E/A-Einheiten oder andere Hardware erforderlich sind.

Die Möglichkeit, Tasks an verschiedene Prozessoren zur Ausführung zu verteilen, führt zu einer sehr wünschenswerten Eigenschaft, nämlich der Softwaretransparenz. Das bedeutet, daß das Anwendungsprogramm unabhängig von der Anzahl der Prozessoren im System wird. Das hat auch zur Folge, daß das vollständige Programm zunächst auf einem einzelnen Prozessor ausgeführt werden kann. Wenn die Ausführungszeit zu lang ist, kann ein zweiter Prozessor – gegebenfalls auch mehr – hinzugefügt werden. Ein ähnlicher Fall tritt ein, wenn das Programm auf N Prozessoren ausgeführt wird und einer davon ausfällt. Dann kann ohne Unterbrechung des Programms die Ausführung weiterlaufen.

Dieser Punkt ist besonders wichtig, denn Softwaretransparenz ist eine Voraussetzung für wirtschaftlich vertretbare Multi-Mikroprozessor-Software.

Einige Spezifikationen des MTOS-68k:

Speicherbedarf in Byte

MTOS/MPOS-Kern	12000
Dateiverwaltung	6150*
Treiber: Konsole	900
Floppy Disk	750*
Zeilendrucker	850*
Overhead/Task	70
Overhead/Device	64

*nicht erforderlich für Echtzeitsteuer-Systeme

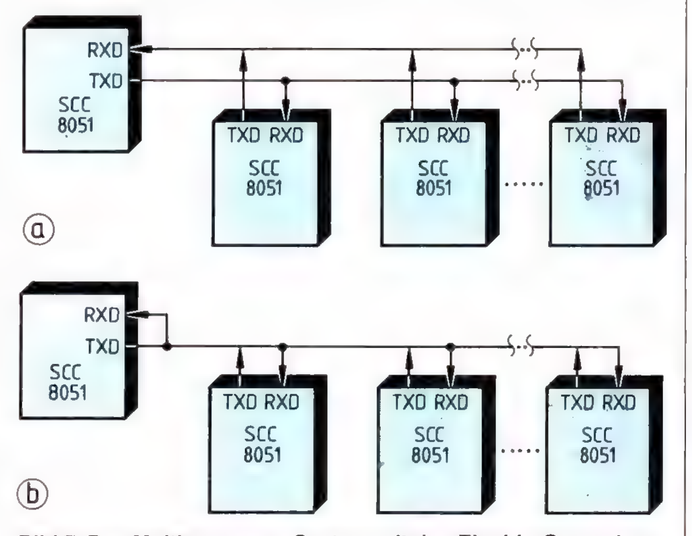
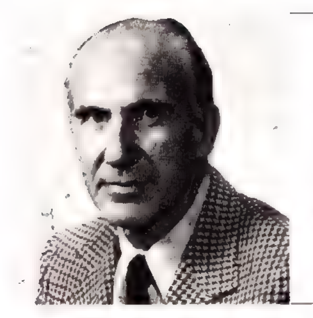


Bild 7. Das Multiprozessor-System mit den Einchip-Computern 8051 kann entweder mit einem Voll-Duplex- (a) oder Halb-Duplex-Bussystem (b) implementiert werden, beide arbeiten mit einer Übertragungsrate von 187,5 kBaud



Hermann Schmid, gebürtiger Kemptner, arbeitet seit seinem Studium in Konstanz (1950) und Syracuse, NY, (MSEE) mit analogen/digitalen Kontrollsystemen, die letzten 20 Jahre bei General Electric in Binghamton, NY. Über sein Spezialgebiet, Mikroprozessoren, unterrichtet er an der Universität.
Hobbys: Bergwandern, Schifahren, Mineraliensammeln und „landscaping".

Weitere Daten

Prioritäts-Ebenen	256
Event-Flags	bis zu 512
Ext. Descretes	bis zu 8000
Device-Typen	bis zu 256
Unit/Device	bis zu 256
Anzahl der Tasks	bis zu 2048
Mailboxes	bis zu 256
Anzahl der µPs	bis zu 16

MTOS-68k ist in 68000-Assemblersprache geschrieben und auf einer 8-Zoll-Diskette gespeichert. Das Basispaket umfaßt den Kern, die MNPS-Option, den Konsolen-Treiber, Debugger, die Diagnoseeinrichtung und ein Demonstrationsprogramm.

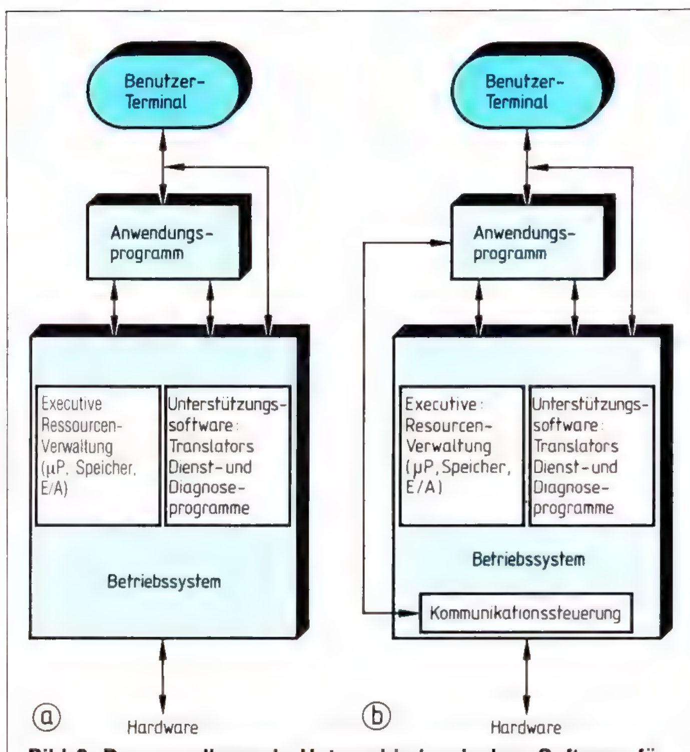


Bild 8. Der grundlegende Unterschied zwischen Software für Einzelprozessoren (a) und Software für Multiprozessoren (b) liegt im Betriebssystem. Dieses muß die lokalen und globalen Ressourcen verwalten sowie die Kommunikation zwischen den Prozessoren steuern

5 Zukünftige Entwicklungen

Jedes Multiprozessorsystem besteht aus Hard- und Software, die zur Systemarchitektur mit dem Systembus, dem Kommunikationsprotokoll sowie einigen Interfaceschaltungen verknüpft sind. Einplatinen- und Einchip-Prozessoren stellen die Hardware, Anwendungsprogramme und Betriebssysteme formen die Software. Derzeit sind noch keine allgemein verwendbaren Multi-Mikroprozessor-Systeme verfügbar, nur einige Hard- und Softwarekomponenten.

Unglücklicherweise sind diese Komponenten von verschiedenen Firmen entwickelt worden. Deshalb ist es nicht ganz einfach, diese zu kombinieren und zu erwarten, daß sie sofort ein arbeitsfähiges System ergeben, auch wenn sie zur gleichen Mikrocomputerfamilie gehören. Aufwendige Anpassungen und Modifikationen sind erforderlich, um ein betriebsfähiges System zu bekommen. Der größte Teil der Entwicklungsarbeit ist daher für die Optimierung und Erhöhung der Effizienz eines Multi-Mikroprozessor-Systems erforderlich.

Es bleibt zu hoffen, daß dieser Zustand sich in der nächsten Zukunft ändern wird, wenn Mikrocomputerhersteller ihre eigenen MPOS entwickelt haben werden oder einige Softwarehäuser für jeden Mikroprozessor ein solches Betriebssystem anbieten. Der 32-Bit-Prozessor iAPX 432 von Intel zeigt die Richtung auf, in die die zukünftige Entwicklung gehen wird. Dieser Typ und seine Architektur in Multiprozessor-Anwendungen werden an anderer Stelle in diesem Heft beschrieben.

Im nächsten Artikel dieser Reihe werden die Leistungsmerkmale des TSCB-Multiprozessorsystems vorgestellt und analysiert, wobei die Zufalls- und Reihentheorie benutzt wird. Eine Computersimulation eines typischen Systems wird ebenfalls dargestellt.

Literatur

[10] *Schmid, H.*: Multi-Mikroprozessorsysteme. Teil 1: Grundlagen. ELEKTRONIK 1982, H. 2, S. 87...95.
[11] *Garrow, R.*: 16-bit single board computer maintains 8-bit family ties. Electronics, October 12, 1978, S. 105...110.
[12] *Nadir, J.*: Bus arbiter streamlines multiprocessor design. Computer Design, June 1980, S. 103...110.
[13] *Denison, A. N.*: Multiprocessing with Z8000-based computers. Electronica, München, 1980.
[14] AmZ8000 User Manual. Advanced Micro Devices, Sunnyvale, Cal., 94086, 1981.
[15] EDN seventh annual µP & µC directory. EDN, November 5, 1980, S. 95...264.
[16] *Hughes, P.*: Single-chip computer aims at multiprocessing. Electronics, October 9, 1980, S. 172...177.
[17] INS8070-series microprocessor family. National Semiconductor Company, Santa Clara, Cal, 95051, April 1979.
[18] MCS-51 family of single-chip microcomputers user manual, Intel Corporation, Santa Clara, Cal. 95051, January 1981.
[19] *Palovski, M.*: Distributed processing using serially linked 8051's. Intel presentation, Intel Corporation, Phoenix Ar., 1980.
[20] *Weitzman, G.*: Distributed micro/minicomputer systems. Prentice Hall Inc. Englewood Cliffs, New Jersey, 07632, 1980.
[21] *Ripps, D. L.*: On operating systems. Collection of relevant articles. Industrial Programming Inc. 100 Jericho Quadrangle, Jericho NY, 11753, 1980.

H. Schmid, R. Naro, A. Gupta

Multi-Mikroprozessor-Systeme

3. Teil: Leistungsmerkmale

In diesem Beitrag sollen die verschiedenen Theorien für die Bestimmung der Wartezeit und der Effizienz eines TSCB-Multi-Mikroprozessor-Systems (*Time-shared common-Bus*) vorgestellt werden, wobei entweder eine Vorrichtung für Vermeidung von Zugriffskonflikten oder eines der verschiedenen Warteschlangen-Konzepte Verwendung findet. Außerdem wird ein Modell zur Computersimulation des TSCB-Systems beschrieben und dessen Ergebnisse mit den theoretischen Resultaten verglichen. Dadurch soll ein tiefer Einblick in alle Faktoren, die die Leistung eines MMPS beeinflussen, gegeben werden, so daß architektonische Gesichtspunkte schon in einem frühen Stadium berücksichtigt werden können.

1. MMPS-Leistung – Herausforderung an den Entwickler

Einer der Vorteile, die in Zusammenhang mit Multi-Mikroprozessor-Systemen immer wieder erwähnt werden, ist das bessere Preis/Leistungs-Verhältnis, das sich im Gegensatz zu Systemen mit nur einem Prozessor erreichen läßt. Dieser Punkt wird aber nur dann nachweislich erfüllt, wenn die Leistungsmerkmale eines MMPS genau definiert, vorausbestimmt und gemessen werden können. Dieses Problem ist allerdings bis heute noch nicht ausreichend gelöst, weil derzeit die leistungsbestimmenden Faktoren eines Systems mit nur einem Prozessor noch nicht einheitlich definiert sind. Das ist allerdings eine Voraussetzung für die eindeutige Bestimmung der MMPS-Leistung. (Siehe auch Kasten „Definitionsmöglichkeiten für die Leistung von Systemen mit einem Prozessor".)

Die Definition der Leistung eines MMPS ist sogar dann problematisch, wenn man voraussetzt, daß die Leistung des einzelnen Prozessors bekannt ist. Das liegt an der starken Abhängigkeit dieses Merkmals von Architektur und Anwendung. Die vielen Parameter, die im einzelnen die Leistung beeinflussen, können in folgende Gruppen eingeteilt werden:

- Die Anzahl und Typen des Datentransfers zwischen den Prozessoren, die wiederum abhängig sind von
  - der Größe des Programms und den Möglichkeiten, wie dieses aufgeteilt werden kann,
  - der Anzahl der globalen Variablen und deren gegenseitigen Abhängigkeit.
- Die Anzahl und die Typen der Prozessoren im System sowie deren Funktionen
  - Mastern, Slaves, Coprozessoren, intelligente Peripherie-Einheiten,
  - auf eine bestimmte Aufgabe zugeschnitten oder mit dynamischer Task-Zuweisung.
- Typ, Bereich und Zugriffsart des Speichers für die globalen Variablen
  - lokaler und globaler Speicher oder beides,
  - Speicher mit einseitigem oder zweiseitigem Zugriff (Dual-Port-Memory),
  - Speicherzugriff: über CPU, DMA oder direkt, z. B. Dual-Port-Memory.
- Die Anzahl, der Typ und die Konfiguration der Systembusse
  - parallel, seriell, Adreß-, Daten-, Steuer-Multiplex usw.,
  - Datenübertragungsrate (Taktfrequenz), Transfer-Kapazität,
  - Typ und Geschwindigkeit des Prioritätscodierers.
- Die Effizienz der Kommunikation zwischen den Prozessoren, bzw. die Verzögerung bei
  - Zugriff auf den Bus, Vergabe und Zuteilung der Steuerung,
  - Auswahl eines Partners durch Aussenden einer Adresse,
  - der Abgabe der Steuerung des Busses durch den Partner,
  - der Adressierung des Speichers des Partners und dem Transfer der Daten.

Hier sind nur die wichtigsten Parameter aufgeführt. Es ist deshalb nicht überraschend, daß eine praktisch anwendbare, aber trotzdem umfassende Methode zur Leistungsbestimmung von MMPS notwendig ist [24...42].

Weil der Grad der Schwierigkeit einer Analyse mit der Komplexität des Systems steigt, soll hier nur auf relativ einfache Systeme eingegangen werden, beispielsweise die Time-Shared-Common-Bus-Architektur, bei der N Prozessoren über einen einzigen Bus miteinander verbunden sind.

2 Leistungsanalyse mit der Zugriffs-Konflikt-Methode

Dieses relativ einfache Verfahren [36] wurde dazu benutzt, die Verzögerung beim Bus-Zugriff (Wartezeit) und die Prozessor-Effizienz eines Systems zu messen, in dem bis zu zehn 8080-CPUs miteinander verbunden sind, und zwar über einen einzelnen Bus und mit einem gemeinsamen Speicher. Folgende Voraussetzungen wurden gemacht:

- Die CPU 0 hat die höchste, die CPU 9 die niedrigste Priorität. Busanforderungen niedrigerer Priorität werden zurückgehalten, bis der Bus frei ist, deshalb ist die CPU 0 eine Effizienz von 100 % und die der anderen entsprechend ihrer Priorität gringer.
- Ein Instruktionszyklus besteht aus verschiedenen Subzyklen, von denen jede wiederum aus vier Maschinenzyklen besteht (t_m), der Bus wird nur während eines der vier t_m-Zyklen benutzt.
- Die Wahrscheinlichkeit p_i, daß die CPU i auf den Bus in einem gegebenen Zyklus zugreift, ist wegen dem Instruktionsformat des 8080 eine Konstante.
- Die CPU-Effizienz kann als $E_i = 1 - d_i$ ausgedrückt werden, wobei d_i die relative Verzögerungszeit ist, die am Prozessor p_i gemessen wurde.
- Die Konflikt-Wahrscheinlichkeit $p_c^{(N)}$ ist eine Funktion von p_i und der Anzahl N der Prozessoren am Bus.

In diesem Fall wurden folgende Schlüsse gezogen: Erstens ist das beschriebene System – verschiedene CPUs greifen auf einen gemeinsamen Speicher zu – ineffizient und wird daher heute auch nicht mehr angewendet. Zweitens wird vorausgesetzt, daß die Zugriffswahrscheinlichkeit für alle CPUs gleich und konstant ist, was allerdings nicht sehr realistisch ist. Drittens sind die mathematischen Beziehungen, mit denen die Zugriffsverzögerungen bestimmt wurden, nicht angegeben, sondern nur die Ergebnisse.

Trotzdem kann das Verfahren als interessante Anregung dienen und für ein TSCB-System ausgeweitet werden. Dann kann es mit dem Verfahren, bei dem die Warteschlangen-Theorie verwendet wird, verglichen werden. In einem TSCB-System ist die Wahrscheinlichkeit von Buskonflikten sowie die Wartezeiten viel kleiner, weil wesentlich weniger Verkehr auf dem Bus

Definitionsmöglichkeiten für die Leistung von Systemen mit einem Prozessor

Obwohl bereits beachtlicher Aufwand in die Lösung dieses Problems investiert wurde, gibt es noch keine einheitliche Methode für die genaue Definition der Leistung von Systemen mit einem Prozessor durch möglichst wenige Parameter. Deshalb ist auch der Leistungsvergleich verschiedener Prozessoren nicht möglich oder führt zu Fehlinterpretationen, weil der wichtigste Parameter – der Durchsatz – von der speziellen Mischung der Instruktionen oder dem Anwendungsprogramm abhängt [30].

Durchsatz ist ein Maß dafür, wieviele Operationen pro s ausgeführt werden, oft in KOPS oder MOPS (tausend bzw. Millionen Operationen pro Sekunde) angegeben. Weil allerdings eine Registeraddition beispielsweise wesentlich weniger Zeit als eine 16-Bit-Multiplikation erfordert, beides aber einzelne Operationen sind, kann kein Vergleich des Durchsatzes auf dieser Basis durchgeführt werden, wenn nicht das Programm oder die einzelnen Instruktionen genau spezifiziert sind.

Ein **Benchmark-Test** ist ein Standard-Programm, mit dem die Leistung eines digitalen Prozessors gemessen werden kann. Wenn alle Anwender und Hersteller sich auf einheitliche Benchmark-Programme einigen würden, wäre das Verfahren wesentlich unproblematischer. Weil aber die Anforderungen der Benutzer höchst unterschiedlich sind (Echtzeit, keine Echtzeit usw.) und weil es im Interesse der Hersteller liegt, die Vorteile ihrer Produkte herauszustellen, wurden so viele unterschiedliche Benchmark-Typen entwickelt. In der letzten Zeit wurden von drei unabhängigen Gruppen verschiedene Benchmarks auf den wichtigsten Prozessortypen (8086, Z8000, 68000 usw.) ausgeführt und miteinander verglichen [31...33]. Die dabei benutzten Programme wurden aus den bekannten *Carnegie-Mellon-Benchmarks* ausgewählt, die die unterschiedlichsten Applikationen einschließen. Die zur Ausführung erforderliche Zeit gibt wahrscheinlich das beste Maß für einen Vergleich des Durchsatzes. Allerdings kann man diesen Ergebnissen nur Bedeutung zumessen, wenn der Prozessor für die gleiche Anwendung eingesetzt wird wie im Benchmark.

Viele **andere wichtige Parameter** werden benutzt, um einzelne Leistungsmerkmale des Prozessors zu charakterisieren: maximaler Speicher-Adreßbereich, Anzahl und Typ der E/A-Kanäle, Verlustleistung, Betriebsmöglichkeit in gestörter Umgebung, Zuverlässigkeit, Abmessungen, Kosten usw. Obwohl diese Parameter für bestimmte Einsatzfälle zugeschnitten sind, sind sie natürlich nicht so sehr applikationsabhängig und können daher im einzelnen sehr leicht verglichen werden.

herrscht. Für die Bestimmung der Parameter kann man dieselben mathematischen Beziehungen verwenden.

3 Leistungsanalyse mit der Warteschlangen-Theorie

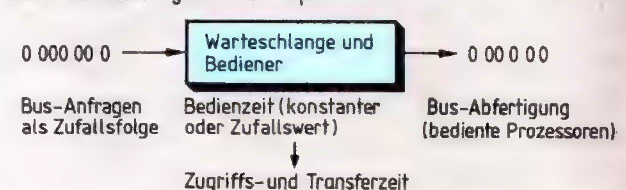
In einem echten TSCB-System (*Bild 1*) kommunizieren die N Prozessoren, bestehend jeweils aus CPU, lokalem Speicher und lokaler E/A-Einheit, über den gemeinsamen Bus. Der Systembus wird im Multiplex-Betrieb benutzt, deshalb kann nur immer einer der N Prozessoren den Bus in Anspruch nehmen. Wenn mehrere Prozessoren gleichzeitig den Zugriff auf den Bus benötigen, kommt es zum Konflikt, der in Übereinstimmung mit den aufgestellten Prioritätsregeln gelöst werden muß (siehe auch Teil 1). Dadurch ergibt sich, daß die Prozessoren geringerer Priorität auf die anderen warten müssen. Der Systembus kann deshalb als eine Warteschlange angesehen werden, in die sich die Prozessoren für den Buszugriff einreihen müssen. Die Zeit, die in der Warteschlange verbracht werden muß, sowie die Dauer, die die Prozessoren an den Bus angeschlossen sind, stellen wichtige Parameter dar, die die Systemeffizienz beeinflussen. Weil jeder Prozessor seinen eigenen lokalen Speicher besitzt und weil nur globale Variable transferiert werden müssen, sind die Datentransfers seltener und die Wartezeiten kürzer als in Systemen, die nur über einen gemeinsamen Speicher verfügen. Die Zeitdauer, während der Prozessor am Bus angeschlossen ist, (die Service-Zeit) hängt von der Datenübertragungs-Kapazität des Busses und dem Überbau an Zugriffs- und Kommunikations-Protokollen, die mit jedem Worttransfer verbunden sind, ab.

Die Verwendung der Warteschlangen-Theorie zur Bestimmung der Leistung eines TSCB-Systems bietet sich an, weil sie bewährte Analyse-Werkzeuge bietet (siehe Kasten „Grundlegende Eigenschaften einer Warteschlange“). Von den verschiedenen alternativen Methoden spiegelt keine ein MMPS ideal wieder. Um mit den Merkmalen und Grenzen der einzelnen Methoden einschätzen zu können, ist es wichtig, mit der statistischen Wahrscheinlichkeit und den exponentiellen Verteilungen vertraut zu sein [37, 38, 39].

Es werden drei alternative Methoden für Warteschleifen beschrieben, die sich in der Anzahl der Benutzer (endlich oder unendlich) und der Art der Service-Rate (exponentielle oder konstante Verteilung) unterscheiden. Die folgenden Voraussetzungen müssen gemacht

Grundlegende Eigenschaften einer Warteschlange

Die grundlegenden Charakteristiken einer Warteschlange werden durch vier Parameter, die in der Gruppe A/B/C/D zusammengefaßt sind, angegeben. Dabei ist „A“ die Verteilung der Anfragen (Beschreibung des Zufallsprozesses), „B“ die Beschreibung des Zufallsprozesses (der Verteilung) der Abfertigung, „C“ die Anzahl der Bediener und „D“ die Anzahl der Prozessoren. Im einzelnen werden die Merkmale von Warteschlangen folgendermaßen dargestellt:
Mittlere Anfragehäufigkeit = λ
Mittlere Service-Zeit = T_s
Maximale Anzahl der Services von einer mittleren Service-Zeit T_s in 1 s = μ



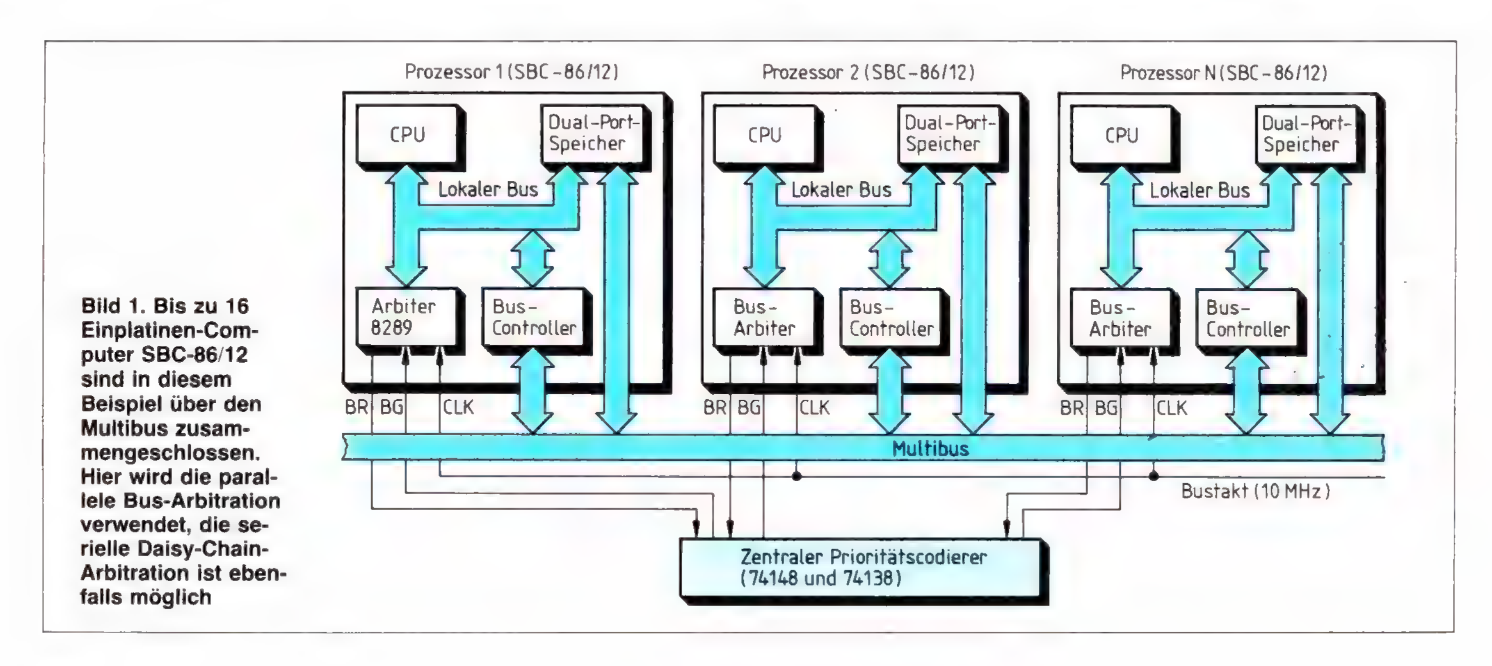


Bild 1. Bis zu 16 Einplatinen-Computer SBC-86/12 sind in diesem Beispiel über den Multibus zusammengeschlossen. Hier wird die parallele Bus-Arbitration verwendet, die serielle Daisy-Chain-Arbitration ist ebenfalls möglich

werden, bevor man sich für die eine oder andere Methode entscheidet:

- Von den K am Bus angeschlossenen Prozessoren sind N im aktiven Zustand (es besteht die Möglichkeit der Busanforderung) und J im blockierten Zustand (entweder in der Warteschlange oder mit dem Bus verbunden).
- Busanforderungen der aktiven Prozessoren treten in zufälliger Reihenfolge auf, wobei die Wahrscheinlichkeit der Dauer zwischen zwei benachbarten Busanforderungen exponentiell mit dem Mittelwert $1/\lambda$ verteilt ist. Alle Prozessoren haben gleiche Anforderungsraten, λ ist die Häufigkeit.
- Die Länge des Datentransfers ist eine Zufallsfunktion und mit dem Parameter $1/\mu$ exponentiell verteilt. Das ist nicht typisch für ein TSCB-System, aber es vereinfacht die Analyse. Die dadurch verursachten Effekte werden später diskutiert.

3.1 *Die M/M/1/∞-Warteschlange*

Diese Warteschlange besitzt eine Anfrage-Häufigkeitsrate nach *Markovian (Poission)*, eine Abfertigungsrate nach *Markovian* (exponentiell), einen einzigen Bediener und eine unendliche Anzahl von Benutzern. Diese Form der Warteschlange kann mit einem einfachen Satz von Bestimmungsgleichungen charakterisiert werden. Die drei Schlüsselparameter (Verkehrsintensität ϱ, Anzahl der Prozessoren im System L, und die Wartezeit W) können sehr einfach mit den zwei Eingangsvariablen λ und μ berechnet werden.
Exponentielle Service-Rate:

$$F(\sigma) = \mu e^{-\mu s}$$

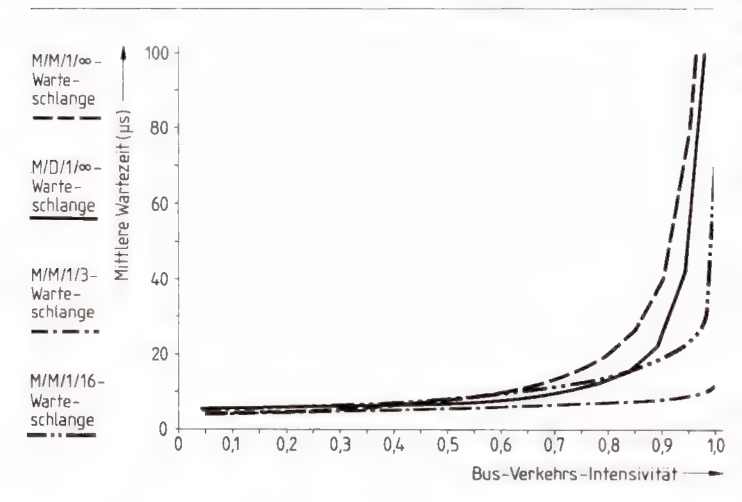


Bild 2. Der Mittelwert der Zeitdauer, die ein Prozessor in einem TSCB-System warten muß, steigt exponentiell mit der Intensivität des Busverkehrs oder der Anzahl der ausgeführten Datentransfers. Der Sättigungseffekt der M/M/1/N-Warteschlange wird durch die Tatsache verursacht, daß mit der wachsenden Zahl der Prozessoren im System (in der Warteschlange oder am Bus) die abnehmende Anzahl der aktiven Prozessoren weniger Busanforderungen verursachen. Das System regelt sich selbst

(Bus-)Verkehrsintensivität:

$$\varrho = \lambda/\mu$$

Mittlere Anzahl der blockierten (wartenden oder gerade bediente) Prozessoren im System:

$$L = \varrho/(1-\varrho)$$

Mittlere Wartezeit im System:

$$W = L/\lambda$$

3.2 *Die M/D/1/∞-Warteschlange*

Diese Warteschlange besitzt eine Anfrage-Häufigkeit nach *Poisson*, eine konstante Service-Zeit und eine unendliche Zahl von Benutzern. Die mathematischen Beziehungen sind ein wenig komplizierter, aber sie benutzen konstante Service-Zeit. Das Gleichungssystem wird mit Hilfe der *Pollaczek-Khintchine*-Formel [40] aufgestellt:
Mittlere Anzahl der blockierten Prozessoren im System:

$$L = \varrho + \varrho^2/2(1-\varrho)$$

Mittlere Wartezeit im System

$$W = L/\lambda$$

3.3 *Die M/M/1/K-Warteschlange*

Diese Warteschlange besitzt Anfrage- und Service-Rate nach *Markovian*, einen Bediener und K Benutzer. Die Gleichungen sind wesentlich komplexer, weil die Zahl der Prozessoren im System endlich ist. Während W in den ersten beiden Warteschlangen eine Funktion von λ und μ ist, ist es bei dieser Warteschlange zusätzlich eine Funktion der Anzahl der Prozessoren am Bus K und von der Anzahl der Prozessoren im System J (wartend oder bedient).
Bus-Verkehrsintensität:

$$\varrho = \lambda/\mu$$

Mittlere Anzahl der blockierten Prozessoren im System:

$$L = \frac{\sum_{i=0}^{i=K} i\,(\varrho)^i \frac{K!}{(K-i)!}}{\sum_{J=0}^{J=K} (\varrho)^J \frac{K!}{(K-J)!}}$$

Mittlere Wartezeit im System:

$$W = L/\lambda.$$

4 Numerische Berechnung der drei Warteschlangen

Mit den hier angegebenen Gleichungen kann die Anzahl der Prozessoren in einem System (L) und die mittlere Wartezeit (W) nun für jede der drei Warteschlangen als Funktion der Eingangsvariablen λ, μ und

N bestimmt werden. Allerdings ist dabei Voraussetzung, daß diese Werte in numerischer Form vorliegen (λ ist die mittlere Häufigkeit der Anfragen, μ die mittlere Servicerate und N die Anzahl der mit dem Bus verbundenen Prozessoren). In diesem Beispiel wurden für N entweder die Werte 3 und 16 gewählt, das sind die maximalen Anzahlen, die jeweils vom seriellen und parallelen Arbitrationsschema des Einplatinencomputers SBC-86/12 unterstützt werden, oder, wenn es für die Warteschlange erforderlich ist, unendlich.

Die Service-Rate μ ist abhängig von Anwendung sowie Architektur und als Kehrwert der mittleren Service-Zeit T_s definiert. Die verschiedenen Warteschlangen besitzen exponentielle, konstante oder Gamma-Verteilung. Um die Berechnung zu vereinfachen und numerische Ergebnisse zu erzielen, wurde eine hypothetische Applikation vorausgesetzt, bei der im Mittel zwei Worte transferiert werden. Für das MMPS SBC-86/12 beträgt die Service-Zeit typisch 4 µs. Daraus folgt, daß die Servicerate μ für die konstante und exponentielle Verteilung 250 000/s beträgt.

Die mittlere Anfragerate λ hängt von Anwendungsprogramm und der Zahl der zu transferierenden globalen Variablen ab. λ muß als unbekannte Eingangsvariable behandelt werden. In Wirklichkeit wird in allen Diagrammen die Bus-Verkehrsintensivität $\varrho = \lambda/\mu$ (anstelle λ) benutzt, um die Abzisse zu normieren und das Problem leichter handhabbar zu machen.

Die Ergebnisse der numerischen Berechnung sind in einer Grafik (*Bild 2*) zusammengefaßt, in der die mittlere Wartezeit für jede Wartschlange als Funktion der Verkehrsintensivität ϱ dargestellt werden.

5 Simulation eines TSCB-Systems

Um die Leistung zu analysieren und einen Bezugswert für den Vergleich der theoretischen Ergebnisse, die im vorhergehenden Abschnitt gewonnen wurden, zu haben, wurde ein Computermodell zur Simulation des TSCB-Systems entwickelt.

Das Multiprozessor-System SBC-86/12 (*Bild 3*) kann, wie bereits in Teil 2 beschrieben, serielle und parallele Arbitration verwenden. Wie in Bild 2 des 2. Teils dargestellt, kann die serielle oder „Daisy-Chain"-Arbitration ohne externe Logik implementiert werden, aber das System ist auf eine maximale Anzahl von drei Prozessoren bei einem Bustakt von 10 MHz beschränkt. Zentrale oder parallele Arbitration (Bild 3 im 2. Teil) erfordert einen kleinen Aufwand an externer Prioritätslogik, aber unterstützt dafür bis zu 16 Bus-Master. Die tatsächliche Implementierung der Prioritätsschaltung hat allerdings wenig Einfluß auf das Konzept des Simulationsmodells.

Der Typ SBC-86/12 ist ein Vielzweck-Einplatinen-Computer mit allen Mechanismen, die für ein Multi-Prozessor-System erforderlich sind. Besonders wichtig sind die 32 KByte Pseudo-Dual-Port-RAM, die 16 KByte Programmspeicher (ROM) und die Multibus-Interface-Schaltungen. Diese sind um den Bus-Arbiter-Baustein 8289 aufgebaut und stellen die für den Multiprozessor-Betrieb erforderliche Arbitrationslogik dar. Das Dual-Port-RAM vereinfacht das Problem des Zugriffs auf den Speicher des Partners.

Das Simulations-Modell ist in PASCAL geschrieben, wobei auf Anpassungsfähigkeit geachtet wurde. Es ist so konzipiert, daß es leicht konfigurierbar ist, so daß ver-

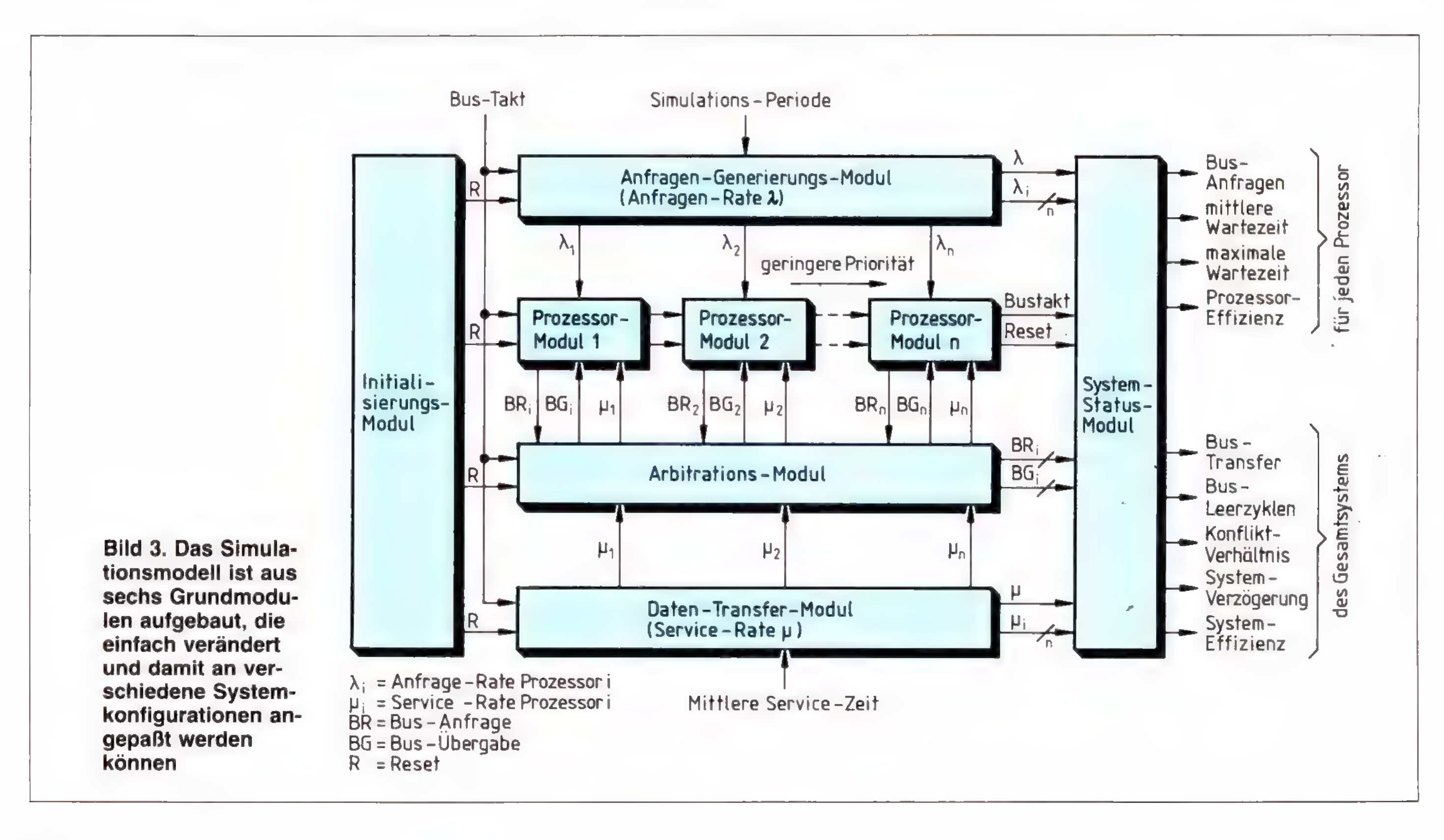


Bild 3. Das Simulationsmodell ist aus sechs Grundmodulen aufgebaut, die einfach verändert und damit an verschiedene Systemkonfigurationen angepaßt werden können

schiedene Systeme mit nur einer einfachen Veränderung der Modellkonstanten simuliert werden können. Diese Konstanten sind zunächst für ein Multibus-System mit SBC-86/12 bestimmt worden.

Während eines jeden Simulations-Zeitintervalls muß das Modell Busanfragen für jeden Prozessor generieren und den Bus den anfragenden Prozessoren zuteilen. Zu jedem Zeitpunkt ist das System in einem der drei folgenden Zuständen: 1. der Bus ist frei, 2. ein einzelner Prozessor will auf den Bus zugreifen, 3. mehr als ein Prozessor wollen auf den Bus zugreifen. Die Arbitrationsroutine untersucht alle Prozessoren und entscheidet, ob einer und welcher Prozessor die Steuerung des Busses übernimmt. Den anfragenden Prozessor mit der höchsten Priorität wird jeweils der Bus zugeteilt. Was aber geschieht, wenn der Bus gerade belegt ist?

Wenn einmal ein Prozessor die Steuerung des Busses übernommen hat, müssen alle anfragenden Prozessoren (unabhängig von der jeweiligen Priorität) warten, bis der derzeitige Bus-Master seine Transaktion beendet hat. Dann erst übernimmt der anfragende Prozessor mit der höchsten Priorität die Steuerung des Busses.

Während jeder Simulationsperiode bringt das Modell den Systemstatus auf den neuesten Stand und speichert die Systemparameter für die spätere Analyse: die Anzahl der Taktperioden, die ein Prozessor bis zur Berücksichtigung wartet, die Anzahl der vorliegenden Busanfragen und die Anzahl der Konflikte zwischen den anfragenden Prozessoren. Am Ende des Simulationsdurchlaufs werden diese Daten benutzt, um die Anzahl der Prozessoren, die Zugriff erlangen wollten, die Wartezeit, bevor der Zugriff möglich ist, und die Gesamteffizienz des Systems zu bestimmen.

Obwohl dieses Modell identische Wahrscheinlichkeiten der Busanfrage für alle Prozessoren voraussetzt, kann dieser Zustand leicht auch für andere Anfrage-Muster modifiziert werden. Die konstante Service-Rate, die in diesem Modell verwendet wird, kann ebenfalls leicht in einen realistischen Wert geändert werden, außerdem auch die Länge der Datenpakete.

Das Computermodell (Bild 3) simuliert das auf dem SBC-86/12 basierende System in allen Aspekten, insbesondere in bezug auf das Bus-Anfrage- und -Zuteilungs-Protokoll. Es umfaßt sechs Grundmodule, die in der folgenden Weise funktionieren:

Das Initialisierungsmodul setzt die betreffenden Parameter während der Einschaltphase des Systems zurück. Das Anfragen-Generierungs-Modul (*Request Generation Module*) erzeugt eine Zufallsrate von exponentiell verteilten Busanfragen während der Interationsperiode und verteilt diese gleichmäßig auf die M Prozessoren in Übereinstimmung mit deren Anfrage-Wahrscheinlichkeit (die in diesem Modell als einheitlich angenommen wird). Die Prozessormodule simulieren die Bus-Anfrage- und Prioritäts-Merkmale des SBC-86/12 und leitet die Anfragen an das Arbitrations-Modul weiter. Diese sind mit dem Simulationstakt synchronisiert und werden gemäß der Priorität zugeteilt. Das Bus-Zuteilungs-Signal (*Bus Grant*) wird an den Prozessor ausgegeben, der die höchste Priorität besitzt. Dieses Signal liegt so lange dort an, bis der Bus wieder frei ist. Das Daten-Transfer-Modul simuliert die Wort- und Blocktransfer-Merkmale des 8086 sowie des Multibusses und erzeugt die mittlere Service-Rate eines jeden Prozessors. Das System-Status-Modul verfolgt die verschiedenen Systemparameter und berechnet diese aus den Simulationsresultaten.

Für jeden Prozessor berechnet das Modell: die Busanfragen, die mittlere und maximale Wartezeit und die Effizienz. Für das Gesamtsystem berechnet das Modell: die Anzahl der durchgeführten Transfers, das Verhältnis der Anfragen-Konflikte, die totale System-Verzögerungszeit sowie die Effizienz des Gesamtsystems. Die Resultate sind in zwei Diagrammen zusammengefaßt (*Bilder 4* und 5), die die mittlere Wartezeit eines jeden Prozessors als Funktion der Bus-Verkehrs-Intensivität für 3- und 16-Prozessor-TSCB-Systeme zeigen.

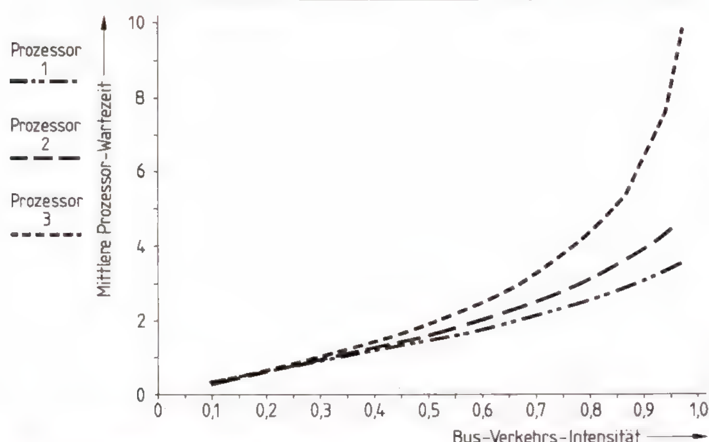


Bild 4. Mittlere Prozessor-Wartezeit eines TSCB-Multiprozessor-Systems mit drei Mastern (SBC-86/12) in Abhängigkeit der Bus-Verkehrsintensivität. Der Betrieb erfolgt in „Locked Operation" und mit 4 μs mittlerer Service-Zeit

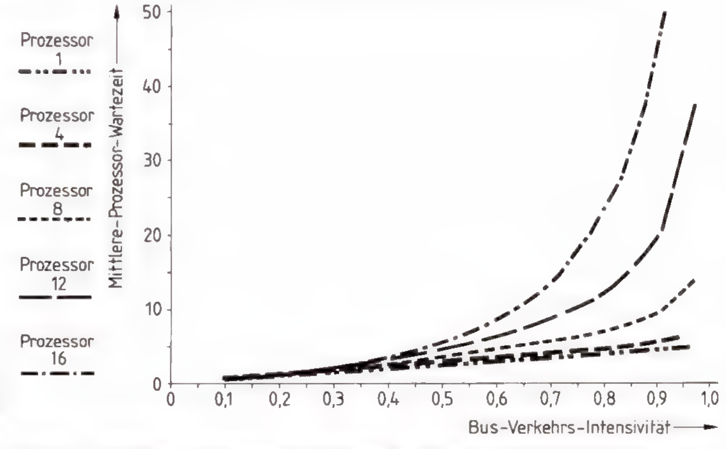


Bild 5. Mittlere Prozessor-Wartezeit eines TSCB-Multiprozessor-Systems mit 16 Mastern (SBC-86/12) in Abhängigkeit der Bus-Verkehrs-Intensivität. Der Betrieb erfolgt in „Locked Operation" und mit 4 μs mittlerer Service-Zeit

6 Vergleich der theoretischen Ergebnisse mit den Simulationsresultaten

Die theoretische Betrachtung und die Simulation bieten wesentliche Einblicke in die Faktoren, die die Leistung eines TSCB-Multiprozessor-Systems bestimmen. Während die theoretische Analyse bewährte Methoden auf das TSCB-System zuschneidet, imitiert die Computersimulation die tatsächlichen arithmetischen und logischen Operationen.

Die Resultate der theoretischen Analyse sind in dem Diagramm, in dem W über ϱ aufgetragen ist (Bild 2), zusammengefaßt. Keine der Warteschlangen ist ideal, weil keine mit priorisierter Arbitration arbeitet, manche setzen eine unendlich große Zahl von Benutzern voraus, andere exponentielle oder konstante Service-Raten. Die ideale Warteschlange wäre vom Typ P/S/1/N, wobei S eine spezielle Verteilung ist, die eine realistische MMPS-Service-Rate wiederspiegelt und priorisierte Arbitration umfaßt. Die dazu notwendigen mathematischen Methoden sind bereits entwickelt, aber noch nicht dokumentiert.

Die Simulationsergebnisse für 3- und 16-Prozessor-TSCB-Systeme (Bilder 4 und 5) zeigen, daß die Wartezeiten W exponentiell mit wachsendem Bus-Verkehr ϱ zunehmen. Wenn ϱ sich dem Wert 1 nähert, wird W für Prozessoren mit geringer Priorität sehr groß (besonders in dem 16-Prozessor-System und das System wird daher sehr uneffizient. Aus diesem Grund sollte es das primäre Ziel eines jeden Systementwurfs sein, die Bus-Verkehrs-Intensivität möglichst weiter unter 1, besser sogar unter 0,7 zu halten. Alle wiedergegebenen Resultate wurden mit einem Modell erzeugt, das mit dem Locked-Bus-Konzept arbeitet, bei dem die Zuteilung während Multi-Wort-Transfers unterbunden ist. Zukünftig sollen Modelle entwickelt werden, die den Bus nach der Übertragung des ersten Wortes wieder freigeben, was für die meisten Anwendungen realistischer ist.

Ein Vergleich der M/M/1/N-Warteschlange mit der Simulation zeigt, daß die Kurven in *Bild* 6 ähnliche Trends aufweisen, aber auch merkliche Divergenzen zeigen. Das läßt sich hauptsächlich auf die Ungenauigkeiten des Modells zurückführen. Das Warteschlangen-Modell umfaßt z. B. keine Priorität, keine realistische Service-Raten-Generierung und/oder eine endliche Anzahl von Benutzern. Das vorliegende Simulationsmodell bietet keine priorisierte Arbitration bei Multi-Wort-Transfers, unterschiedliche Markovian-Anfrage-Muster für jeden Prozessor und eine realistische Service-Raten-Generierung. Trotzdem ist das Simulationsmodell akkurater und wird deshalb als Referenzkurve benutzt.

Im einzelnen sind die Resultate nur für Verkehrs-Intensivitäten von 0,00 bis 0,90 gezeigt, weil beide Modelle bei gesättigtem Bus starke Abweichungen zeigen. Der Grund dafür liegt in der Tatsache, daß die Zahl der wartenden Prozessoren stark ansteigt, ohne daß gleichzeitig der Verkehr auf dem Bus weiter zunehmen kann. Das wiederum führt zu einer Verlängerung der

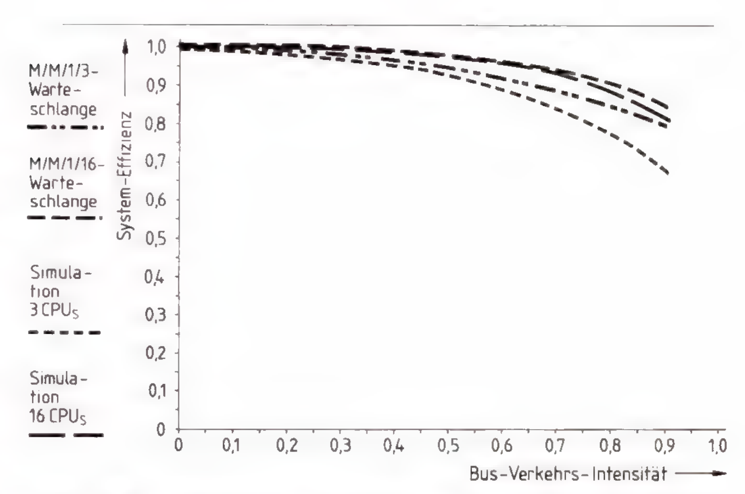


Bild 6. Die verschiedenen Resultate der theoretischen Untersuchung und der Simulation zeigen gute Übereinstimmung. Unterschiede ergeben sich durch die Ungenauigkeiten der verwendeten Methoden

Zeit, die ein Prozessor in einer Warteschleife zubringt sowie zu einer Abnahme der Systemeffizienz.

Man sollte beachten, daß die Systemeffizienz mit der Anzahl der Prozessoren zunimmt. Das liegt daran, daß jeder Prozessor in gleichem Maße zur gesamten Bus-Verkehrs-Intensität beiträgt und das Hinzufügen weiterer Prozessoren die Anfragen-Rate nicht verändert. Daher ist es weniger wahrscheinlich, daß der einzelne Prozessor vom Buszugriff ausgeschlossen ist, wodurch die Effizienz des Gesamtsystems steigt. Das ist allerdings nicht der Fall, wenn die zusätzlichen Prozessoren auch die gesamte Arbeitsbelastung erhöhen. In diesem Fall, der hier nicht näher erläutert werden soll, nimmt die Effizienz des Gesamtsystems ab, weil die Anfrage-Rate und die Bus-Belegungen zunehmen, wenn die Zahl der Prozessoren steigt.

Literatur

[30] *Kobayashi, H.*: Modeling and Analysis: Introduktion to Performance Analysis. Addison-Wesley Publishing Company, Reading, MA, 1978.
[31] *Grappel, R. D. et al*: A Tale of four µPs: Benchmark Quantify Performance. EDN 1981, 1. April, S. 179...285.
[32] *Blacksburg*: 16-bit Mikroprocessors. Howard W. Sams & Co., 1981.
[33] *Nelson, V. P.*: Comparison of 16-bit Microcomputers. IEEE Micro 1981, H. 1.
[34] *Hoogendorn, C. H.*: A General Model for Memory Interference in Multiprocessors. IEEE Transactions on Computers, Oktober 1977, S. 998...1005.
[35] *Bhandarkar, D. P.*: Some Performance Issues in Multiprocessing Design. IEEE Transaction on Computers, Mai 1977, S. 506...510.
[36] *Hoerner, S.*: Efficiency of a Multi-Microprocessor System with Time-shared Buses. Euromicro 1977 Proceedings, Amsterdam, S. 35...42.
[37] *Gross, d. et al*: Fundamentals of Queueing Theory. John Wiley & Sons, New York, 1974.
[38] *Graham, G. S.*: Queueing Network Models Computer System Performance. Computing Surveys Volume 10, H. 3, September 1978, S. 219...224.
[39] *Wiley, J. M*: Just Enough Queueing Theory. Datamation, Februar 1977, S. 87...96.
[40] *Bibbero, R. J.*: Simulating a Control System's Data Highway. Control Engineering, Juli 1981, S. 81...83.
[41] *Marsan, M. A. et al*: Analysis of the Communication Network in Multi-Microprocessor Systems. Euromicro 1979 Proceedings, S. 381...390.
[42] *Marsan, M. A. et al*: Performance Evaluation of the u* Multiprocessor System. Proceeding of the IEEE Juni 1979, S. 106...115.

H. Schmid, R. Naro, G. Franzkowiak

Multi-Mikroprozessor-Systeme

4. Teil: Entwicklungssysteme für Mehrprozessor-Konfigurationen

Betrachtet man alleine die Hardware, sind die derzeitig verfügbaren Multi-Mikroprozessor-Systeme für die Realisierung kosteneffektiver Lösungen bei vielen Applikationen geeignet. Es sind bereits eine große Zahl Einplatinen- sowie Einchip-Computer erhältlich, die ein weites Leistungsspektrum bei geringen Kosten bieten. Die verschiedenen Systembusse, z. B. VME-, S100- und Multi-Bus, unterstützen Multi-Master-Protokolle und vereinfachen die Implementierung von Multiprozessor-Systemen. Wie aber steht es mit der Entwicklung von Anwendungs-Software von Mehrfachprozessoren, dem dazugehörigen Hardware-Debugging sowie der Hardware-Software-Integration? Nach welchen Gesichtspunkten kauft oder entwickelt man ein Betriebssystem, das in der Lage ist, Multi-Tasking-Operationen auf Mehr-Prozessor-Systemen zu unterstützen, diese Tasks nach einem Zeitplan ablaufen zu lassen, die gemeinsamen Ressourcen zu verwalten, die Kommunikation zwischen den Prozessoren zu synchronisieren, mit der Umgebung in Verbindung zu treten sowie Daten mit peripheren Geräten, wie Terminals, Drucker und Massenspeicher, auszutauschen?

Die sich ausweitende Anwendung von Mikrocomputern und die Notwendigkeit, Hard- und Softwareentwicklung zu vereinfachen, führte dazu, daß sich ein vollständig neuer Markt für Mikroprozessor-Entwicklungswerkzeuge entstand. Es handelt sich im einzelnen um:

- konventionelle Testgeräte wie Oszilloskope, Logik-Analysatoren, Puls-/Takt-/Wortgeneraturen usw.
- spezielle Test- und Programmiergeräte wie Takt-/Speicher-/ Mikrocode-Simulatoren, PROM-Programmierer usw.
- konventionelle Mikrocomputer-Entwicklungssysteme wie die „Intellec"-Serie (Intel), „EXORciser" (Motorola) usw.
- spezielle Mikrocomputer-Entwicklungssysteme wie „Microscope" (Intel), „Serie" 1000 (Millenium) usw.
- spezielle Hard- und Software-Debugging-Einrichtungen wie In-Circuit-Emulatoren usw.

Am weitesten verbreitet sind heute konventionelle Mikrocomputer-Entwicklungssysteme (MDS – Microcomputer Development Systems), allerdings unterstützen diese Geräte in der Regel nur einzelne Prozessoren und einen Benutzer. Erst in letzter Zeit kommen Systeme auf den Markt, die sich für den Mehr-Benutzer-Betrieb eignen. Das „Z-Lab" von Zilog ist ein Beispiel dafür. Aber welche Geräte unterstützen die Entwicklung von Systemen mit mehreren Prozessoren?

Zu den wenigen bislang auf dem Markt erhältlichen Einrichtungen zur Entwicklung von Multi-Prozessor-Systemen gehört der „Multi-ICE 2302" von GenRad und das System 9516 von Millenium [43, 44]. Beide bieten umfangreiche Möglichkeiten zur Echtzeit-Hardware-Emulation sowie Hardware-/Software-Integration bei bereits aufgebauten Multiprozessor-Systemen (siehe Kasten: *Industriell gefertigte Multi-Mikroprozessor-Entwicklungssysteme*).

Nicht angeboten werden derzeit Geräte, die es erlauben, Emulation und Echtzeit-Evaluation im frühen Stadium der Entwicklung durchzuführen. Dabei könnten frühzeitig unterschiedliche Prozessor-Architekturen und System-Softwarepakete auf ihre Eignung für eine spezielle Anwendung überprüft werden. Außerdem vereinfacht sich Entwurf, Codieren, Testen und Debugging parallel ablaufender Applikationsprogramme. Weil von der Industrie nach Kenntnis der Autoren kein derartiges System angeboten wird, modifizierten sie ein konventionelles MDS, so daß es die genannten Anforderungen erfüllt. Die gesamte Hardware und der größte Teil der Software können mit relativ geringem Kostenaufwand (etwa 12 000 $ in den USA) erworben werden. Lediglich die relativ einfache System-Steuersoftware muß geschrieben werden.

Dieser Beitrag soll insbesondere dazu dienen,

- die Notwendigkeit der Hard- und Software-Werkzeuge zur Unterstützung *aller* Phasen der Multi-Prozessor-Entwicklung zu zeigen
- zu verdeutlichen, daß Entwicklungseinrichtungen wie der „Multi-ICE" von GenRad oder das System 9516

Industriell gefertigte Multi-Mikroprozessor-Entwicklungssysteme

Sowohl GenRad als auch Millenium haben ihren existierenden Entwicklungssystemen für Einzelprozessoren Einrichtungen zur Multi-Mikroprozessor-Entwicklung hinzugefügt. In erster Linie sind das die verschiedenen In-Circuit-Emulatoren (ICE), die über den Systembus mit dem Hostcomputer verbunden sind (*Bild X*). Jede ICE-Einheit hat Karten, auf denen sich zusätzliche Speicher, Emulations- und Debug-Funktionen, Echtzeit-Tracing (Logik-Analysator) sowie Ereignis-Detektion befinden. Eine „Personality-Probe" verbindet die Emulator-Schaltung in jeder ICE-Einheit mit der Platine des Zielsystems. Die Software-Entwicklung erfolgt in herkömmlicher Weise auf dem Hostcomputer des Grundsystems.

Die Systeme von GenRad und Millenium bieten wirtschaftliche und wirkungsvolle Unterstützung beim Hardware-Debug und der Hardware/Software-Integration (*Tabelle*) – allerdings nur bei Multiprozessor-Systemen, die bereits aufgebaut sind. Werkzeuge zur Unterstützung der funktionellen Entwicklung eines Multi-Mikroprozessor-Systems, z. B. Aufteilung des Benutzerprogramms in konkurrierende Tasks, Evaluation der Interprozessor-Kommunikation oder der Leistungsmerkmale alternativer Architekturen, bieten diese Systeme nicht.

Neben denbeiden bereits erwähnten Produkten ist inzwischen ein Entwicklungssystem von Philips auf den Markt gekommen, das ähnliche Eigenschaften wie die Geräte von Millenium sowie GenRad zeigt [53].

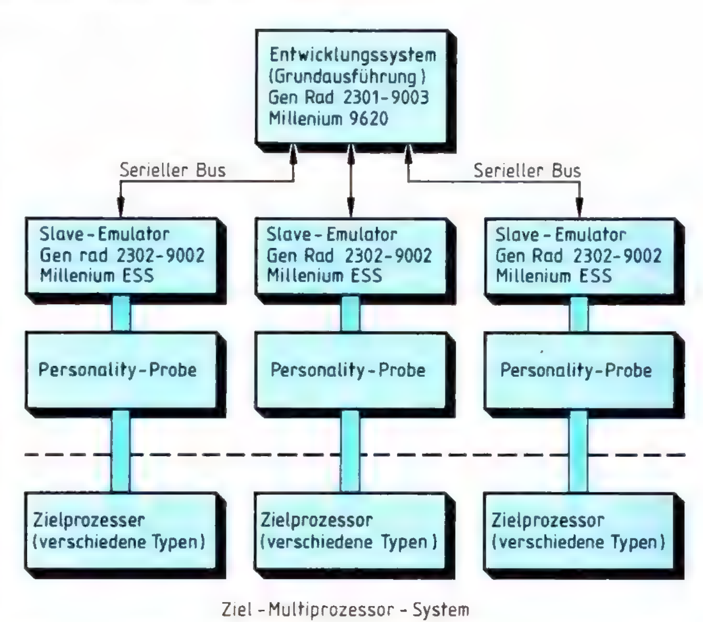


Bild X. An den Hostcomputer können mehrere ICE-Einheiten angeschlossen werden. Ein Anschlußmodul von jeder Einheit ist mit jeweils einer Prozessor-Karte verbunden. Der Hostcomputer verwaltet und steuert die Systemoperation, das Benutzer-Interface und die Anzeige von Daten

Systeme für die Multi-Prozessor-Entwicklung

Merkmale	System 2302 (GenRad)	9516 (Millenium)
Unterstützte Architekturen	viele Multiprozessor-Systeme	viele Multi-Prozessor-Systeme
Unterstützte Prozessoren	die meisten µPs	die meisten µPs
Anzahl der µPs im System	bis zu vier	bis zu vier
System-Hardware	Inform.-Steuer-System: 17 k $	System 95SP16: 21,5 k $
– Hostcomputer und Speicher	Z80, 64 K RAM, 4 K ROM	Z80, 40 K RAM, 4 K ROM
– Terminal	CRT mit 24 x 80 Zeichen	VT-100 (24 x 80 Zeichen)
– Massenspeicher	8-Zoll-Doppel-Floppy	8-Zoll-Doppel-Floppy
– MDS/Emulator-Verbindung	(seriell ?)	seriell
System-Software		
– Betriebssystem	UDOS (GenRad)	M/PM-Multi-Tasking
– Cross-Assembler	ja	ja
– Linker, Debugger	ja	ja
– HLL-Compiler	PASCAL-G (2302-9807)	PC8080-PASCAL
In-Circuit-Emulator (ICE)	Slave-Emulator [1]	Slave-Emulator [2]
– Zahl der erf. ICE	einer pro Zielprozessor	einer pro Zielprozessor
– Emulation	bis 6 MHz, transparent	bis 8 MHz, transparent
– Emulator-RAM	32 K/80 ns	32 K/100 ns
– Logik-Trace-Umfang	256 Ereignisse/64 Kanäle	2 x 1024 Ereign./80 Kanäle
– Hardware-Breakpoints	4 verschachtelt u. bedingt	einfach/komplex/super/global

Bemerkungen

1) Die Konfiguration besteht aus Slave-Emulator 2303-9002, Prozessor, Interface, Analysator mit 32 K RAM und Personality-Probe sowie Software für Z8000.
2) Die Konfiguration besteht aus XS-Emulator (und Pod), Debugger, Analysator (und Tastköpfe) sowie 32-K-Speicherkarten für Z8000.
3) Hardware-Breakpoints (Adressen- und Steuersignale) mit Verschachtelung und verschiedenen Bedingungen wie z. B. Speicher- und E/A-Bereiche, Ereignis-Zählung und externe Ereignisse.
4) einfach: Adressen und Operations-Typ; komplex: Adresse, Status, Daten, externe Ereignisse und Bedingungen; super: sequentielle Kombinationen einfacher und komplexer Ereignisse; global: jedes einfache, komplexe und globale Ereignis auf vier globalen Leitungen.



Bild Y. Das Multi-ICE-Entwicklungssystem von GenRad umfaßt die System-Konsole, ein Doppel-Floppy-Disk-Controller (2301-9003) und bis zu vier Slave-Emulator-Einheiten (2301-9001). In jeder ist ein Emulator-Board, eine Speicherkarte und eine Platine mit dem Logik-Analysator (optionell). Mit dem Emulator ist z. B. die Personality-Probe (2302-9011) für den Z8002 verbunden, die in die CPU-Fassung des Zielprozessors eingesteckt wird.
(Foto: GenRad/Future Data/Kontron)

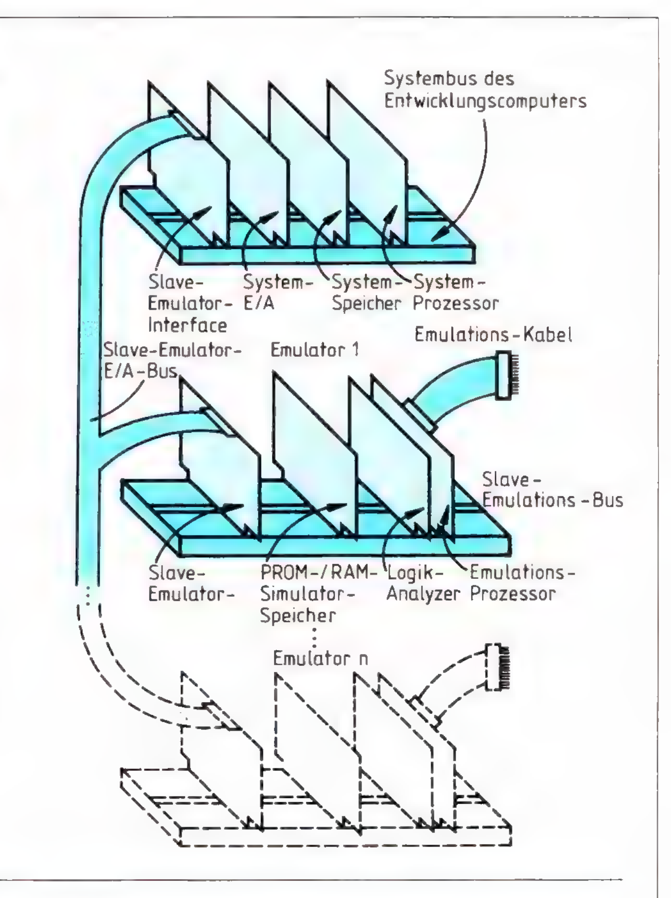
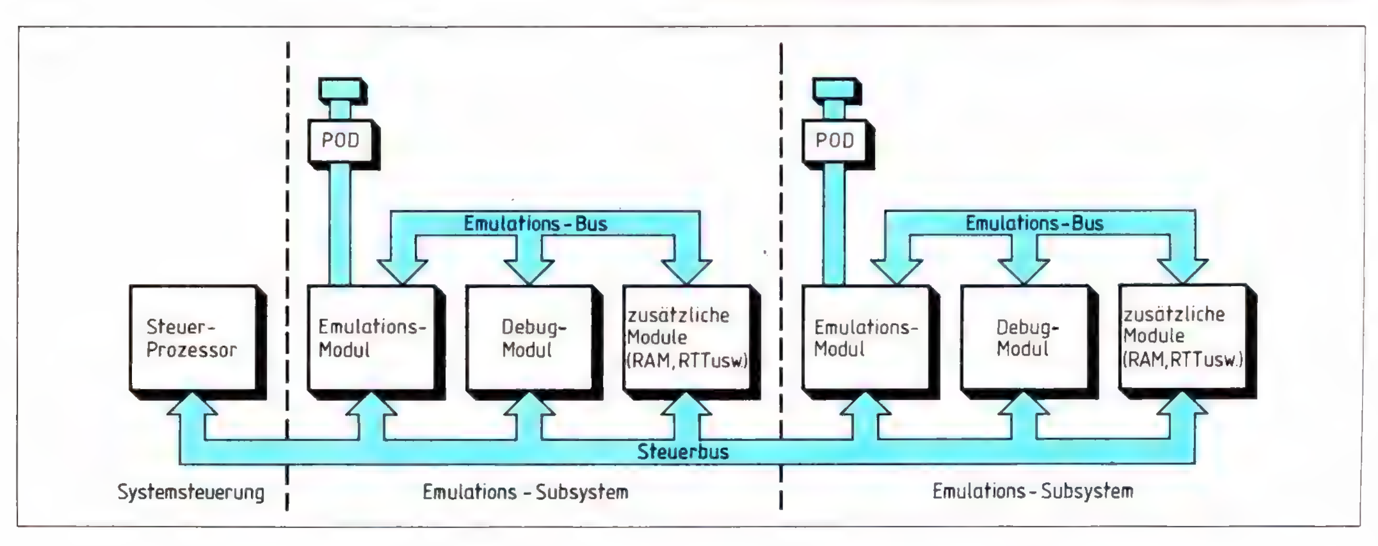


Bild Z. Das System 9516 von Millenium besitzt ein Terminal, das dem VT-100 ähnelt, eine Station 9520 mit zwei 8-Zoll-Floppy-Disk-Laufwerken. Bis zu vier Emulations-Subsysteme (ESS) können an den Bus des Hostcomputers angeschlossen werden (zwei direkt, zwei weitere an die Erweiterungs-Box). Jedes ESS verfügt über ein Emulator-Board, eine Debug-Platine mit 8 K RAM, eine optionelle Echtzeit-Trace-Einheit (RTT) und eine Ereignis-Detektierungs-Karte. Außerdem bietet das System 9516 benutzerprogrammierbare Tasten, Menue-Anzeige des Handbuchs, einen Kommando-Zeilen-Prozessor für schnelle und direkte Eingabe (Foto: Millenium/Spezial-Elektronik)





von Millenium zwar leistungsfähige Unterstützung für bereits aufgebaute Prozessor-Konfigurationen bieten, die anfängliche Entwicklungsphase aber vernachlässigen

- die Notwendigkeit für ein Entwicklungssystem gerade auch in diesem Projektabschnitt nachzuweisen

- zu zeigen, wie ein existierendes SMDS in ein MMDS umgebaut wird

- den Bedarf an vollständiger Multiprozessor-System-Unterstützung aufzuzeigen, der von der betreffenden Industrie noch nicht befriedigt wird.

1 Merkmale derzeit verfügbarer Entwicklungssysteme für Einzelprozessoren (SMDS)

Entwicklungssysteme für Einzelprozessoren (SMDS – Single Microprocessor Development System) werden sowohl von Halbleiter-Herstellern (Intel, Motorola usw.) als auch von Meßgeräteproduzenten angeboten (Hewlett-Packard, Tektronix, Philips usw.). Geräte der Halbleiterhersteller unterstützen normalerweise nur deren Produkte, während die anderen Systeme für viele Mikroprozessortypen geeignet sind [45...52].

Ein typisches SMDS für 8- und 16-Bit-Mikroprozessoren besteht aus einen 8- oder 16-Bit-Hostcomputer, einem Doppel-Floppy-Disk-Laufwerk, einem Drucker, einem Ein-Ausgabe-Terminal mit Tastatur und Bildschirm. Die erforderliche Standard-Software umfaßt Dienst- und Diagnose-Programme, Assembler, Editoren, Binder usw. Compiler für Hochsprachen sind meistens als Optionen erhältlich. Die gesamte Software läuft auf dem Hostcomputer. In-Circuit-Emulatoren sind Zubehör. Ein SMDS führt in der Regel folgende Funktionen aus:

- Entwicklung von Anwendungssoftware auf dem Hostcomputer
  - Schreiben, Editieren, Assemblieren, Binden, Testen und Debugging von Benutzerprogrammen
  - Ausführung des kompletten Programms, eines bestimmten Teiles davon oder einer Zeile
  - Anhalten beim Auftreten eines Fehlers, Anzeige des Quellen-/Maschinen-Codes, Modifizierung und erneutes Ausführen

- Hardware-Test und -Debugging mit dem ICE
  - Ausführung eines Benchmarks auf dem Zielprozessor, im Ziel- oder Host-Speicher sowie den E/A-Einheiten
  - Anzeige und Modifizierung des Inhaltes von Registern, Stack-/Speicher-Plätzen
  - Ausführung von Steuer-/Monitor-Code mit dem Analysator, der Breakpoints setzt

- Hardware-/Software-Integration mit dem ICE
  - Transfer des Anwendungsprogramms vom Massenspeicher zum RAM
  - Reset und Initialisierung der Zähler, Zeitgeber, Flags usw.
  - Ausführung des Programms Zeile für Zeile und Überwachung der Daten-/Steuerungs-Transfers
  - Trace-Programm-Ausführung zum Test auf Fehler, bei denen angehalten wird
  - Aktivierung des Interrupts, Überwachung und Speichern des Status, Sprung in das Unterprogramm

- Firmware-Generierung
  - Programmierung der Speicherbausteine (PROM, EPROM).



Bild 1. Ein typisches Entwicklungssystem für Einzel-Prozessoren und Ein-Benutzer-Konfigurationen ist der Typ AMD SYS-8/8 (Foto: AMD)

2 Anforderungen an zukünftige Multi-Mikroprozessor-Entwicklungssysteme (MMDS)

Multi-Mikroprozessor-Systeme werden sich erst dann auf breiter Front durchsetzen, wenn die erforderlichen Werkzeuge zur Verfügung stehen, die wenigstens dem derzeitigen Umfang der Entwicklungsunterstützung bei Einzelprozessoren entsprechen. Das bedeutet, daß ein MMDS auch in der Lage sein muß, alle Entwicklungsphasen zu unterstützen. Im einzelnen sind diese Phasen:

- Entwicklung der Anwendungs-Software
  - Aufteilung der Programme in Concurrent-Tasks, so daß
    - die Parallel-Verarbeitungs- und Intertask-Kommunikation optimiert wird
    - die Tasks die Interface-Standards erfüllen.
  - Schreiben, Assemblieren, Binden, Testen, Debugging und Verifizierung von Mehrfach-Tasks-Programmen,
    - die sequentiell auf einem oder parellel auf N Prozessoren ausgeführt werden
    - die permanent spezifischen Prozessoren zugeordnet sind (im ROM gespeichert)
    - die während der Laufzeit dynamisch einem freien Prozessor übergeben werden
    - die durch Semaphore, Flags, Messages usw. gesteuert und synchronisiert werden
- Evaluation, Simulation und Emulation des Zielsystems
  - Evaluation verschiedener Zielarchitekturen als Alternativlösungen in bezug auf
    - Gesamtleitung, also Durchsatz, Effizienz, Verzögerungen usw.
    - Effizienz der Kommunikations- und Arbitrations-Techniken
    - mögliche Fehlfunktionen und Rekonfigurations-Pfade
    - Ausführung eines Benchmark-Programms auf dem Emulator in Echtzeit
  - Ausführung jeder Anwendungs-Task und des kompletten Programms auf
    - dem Software-Simulator des Zielsystems
    - dem Hardware-Emulator des Zielsystems
  - Emulieren der Systemoperationen für den Zielprozessor, z. B.
    - Starten und Stoppen der Task-Ausführung
    - Herunterladen der Benutzertasks auf einen spezifischen Prozessor
    - Überwachung der Operation, Anhalten bei Auftreten eines Fehlers und Anzeige der wichtigen Daten
- Hardware-Debugging und Hard-/Software-Integration
  - Verfolgung jeder Zuweisung eines Prozessors oder des gemeinsamen Speichers
    - Emulation des Speichers von 1...N Zielprozessoren mit System-Speicher
    - Abbilden des System- oder Emulator-Speichers im Zielspeicher
  - Ausführung einer oder N Tasks auf Emulator-Prozessor(en) der entwickelten Hardware-Umgebung in Form
    - kompletter Task(s) oder als Satz konkurrierender Instruktionen
    - bzw. auf spezifischen Prozessoren oder in dynamischer Dispatch-Betriebsart
  - Anhalten beim Auftreten eines Fehlers, Anzeige der laufenden Instruktion(en) und
    - Identifizierung der fehlerhaften Task und/oder des Prozessors, der diese ausführt
    - Vorsehen von Einrichtungen zur Modifizierung des Codes und Restart des Systems
  - Ausführung der Benutzer-Tasks in Concurrend-Single-Step-Methode durch
    - Einzelschrittbetrieb eines Prozessors, während die anderen angehalten werden
    - Einzelschrittbetrieb aller Prozessoren gleichzeitig durch Ablaufenlassen einzelner Takt-/Speicher-/Maschinen-Zyklen
  - Überwachung der Multiprozessor-Operationen bei normaler und Einzelschritt-Ausführung durch Lesen, Anzeigen und Modifizieren
    - der Global-Daten im zentralen oder verteilten gemeinsamen Speicher
    - Adressen und Daten auf dem gemeinsamen Bus im Einzelschrittbetrieb
    - Semaphore oder Messages in den Mailboxes

3 Wie man aus einem konventionellen SMDS ein MMDS macht

Sowohl der „Multi-ICE" (GenRad) als auch das System 2302 (Millenium) bieten leistungsfähige Hardware-Emulation und unterstützen wirkungsvoll die Hardware-/Software-Integration. Allerdings können diese Geräte, wie bereits erwähnt, nicht für die anfängliche Phase des Multiprozessor-Entwurfs und der -Evaluation verwendet werden. Die Unterstützung der Entwicklung von Anwendungssoftware, die von diesen Geräten geboten wird, beschränkt sich lediglich auf Systeme mit einem Prozessor. Außerdem können sie Multiprozessor-Architekturen vor der eigentlichen Implementierung nicht emulieren, simulieren und evaluieren.

Gerade für die Anfangsphase des Multiprozessor-Entwurfs und der -Emulation stehen keine Werkzeuge zur Verfügung. Aus diesem Grund wurde bei General Electric (USA) ein MMDS implementiert, das folgende Aufgaben erfüllen soll:

1. Unterstützung der Software-Entwicklung
2. Evaluation einer Echtzeit-Architektur
3. Emulation eines Multiprozessors-Systems mit gemeinsamem Bus, der im Multiplex-Betrieb benutzt wird

(Siehe auch 2. Teil dieser Aufsatzreihe). Dieses System ist ähnlich wie das Ziel-System aufgebaut, aber mit

fertig käuflicher Hardware implementiert. Das vorgeschlagene MMDS hat nicht die Möglichkeit des Multi-ICE-Systems und kann daher weder die Hardware-Debugging- noch die Hardware-/Software-Integrations-Merkmale erfüllen, die weiter oben zusammengestellt sind. Allerdings ist es damit möglich, preiswert und schnell Software- und Hardwareentwicklung für Multiprozessoren auszuführen, insbesondere in den frühen Evaluations- und Hardware-Entwurfsphasen. Vorteil des vorgeschlagenen MMDS ist, daß es durch Hinzufügen verschiedener Einplatinen-Computer, die dem Ziel-System ähnlich sind, und System-Steuersoftware zu einem konventionellen SMDS mit Standard-Systembus und freien Einschubmöglichkeiten im Chassis realisiert werden kann. In den meisten Fällen ist ein solches System bereits bein Anwender vorhanden.

Die meisten Entwicklungsarbeiten für Multiprozessor-Anwendungssoftware können mit den Standard-Programmpaketen des SMDS ausgeführt werden, besonders dann, wenn man eine Hochsprache benutzt. Das wichtigste Merkmal des vorgeschlagenen MMDS ist hingegen die Fähigkeit, eine Emulation des Zielsystems auf käuflich verfügbarer Hardware durchführen zu können, bevor die Hardware des Anwenders konstruiert wurde. Multitask-Benutzerprogramme können beispielsweise in Echtzeit auf dem Zielcomputer ausgeführt werden, indem andere Techniken zum Transfer von Daten zwischen den Prozessoren und/oder zur Arbitration, ein zentraler oder verteilter gemeinsamer Speicher usw. Verwendung finden. Auch können verschiedene Zielprozessoren in einer realistischen Multiprozessor-Umgebung durch Ausführung von Benchmark-Programmen untersucht werden.

Das vorgeschlagene MMDS ist mit dem konventionellen SMDS SYS-8/8 von AMD implementiert. Die Grundausführung umfaßt einen Einplatinen-Hostcomputer auf der Basis der CPU 8080, einen 64-KByte-Schreib-Lese-Speicher sowie einen Floppy-Disk-Controller. Diese Systemgruppen sind über einen Multibus miteinander verbunden (*Bild 2*). Im Chassis des SYS-8/8 sind vier der insgesamt sieben Einschübe für Erweiterungen freigelassen. Drei davon werden zum Aufbau eines Multiprozessor-Systems auf Basis der CPU Z8000 benutzt. Dazu werden die Multibus kompatiblen Einplatinen-Computer AM-96/4116 eingeschoben. Durch diese Erweiterung wird das SMDS zu einem MMDS. Außer wenigen Vorkehrungen (erstmaliges Setzen der Schalter und Drahtbrücken zur Festlegung der Adreßbereiche usw.) sind keine Hardware-Modifikationen notwendig. Ähnliches trifft für die Software-Unterstützungspakete zu. Erforderlich ist allerdings neue Software für die Systemsteuerung des Hostcomputers. Außerdem muß der Monitor des Zielprozessors umfangreichen Modifikationen unterzogen werden.

4 Software: Schlüssel zur Systemmodifikation

Das Hinzufügen mehrerer Ziel-Prozessoren in Form der Einplatinen-Computer zum konventionellen Entwicklungssystem führt zu einer Multi-Prozessor-Konfiguration (*Bild 3*), bei der der Multibus die Verknüpfung übernimmt. In einem solchen System sind ein zentraler gemeinsamer Speicher (64-Kbyte-RAM-Karte) sowie verteilte Speicher (32 KByte Dual-Port-RAM, auf jeder

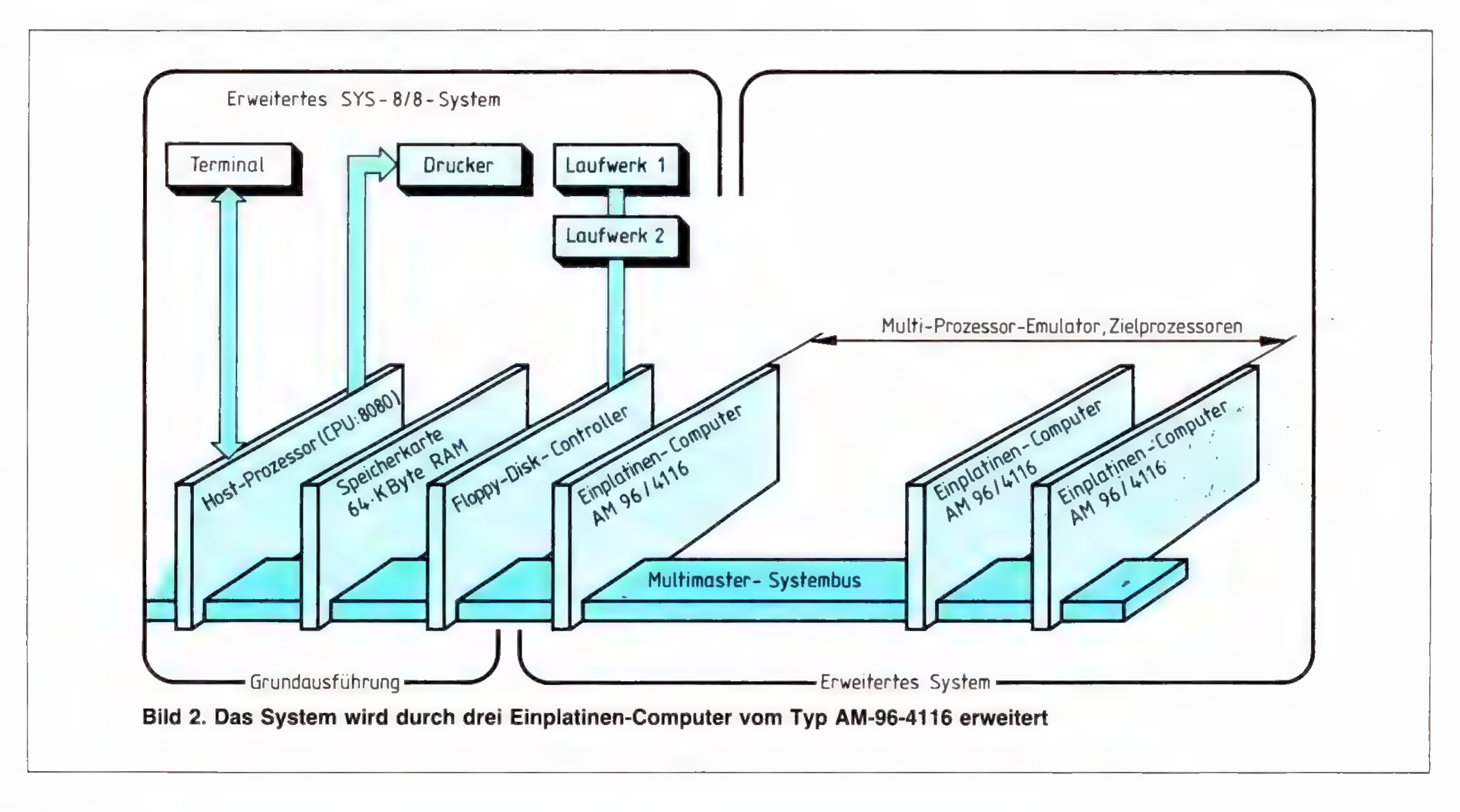


Bild 2. Das System wird durch drei Einplatinen-Computer vom Typ AM-96-4116 erweitert

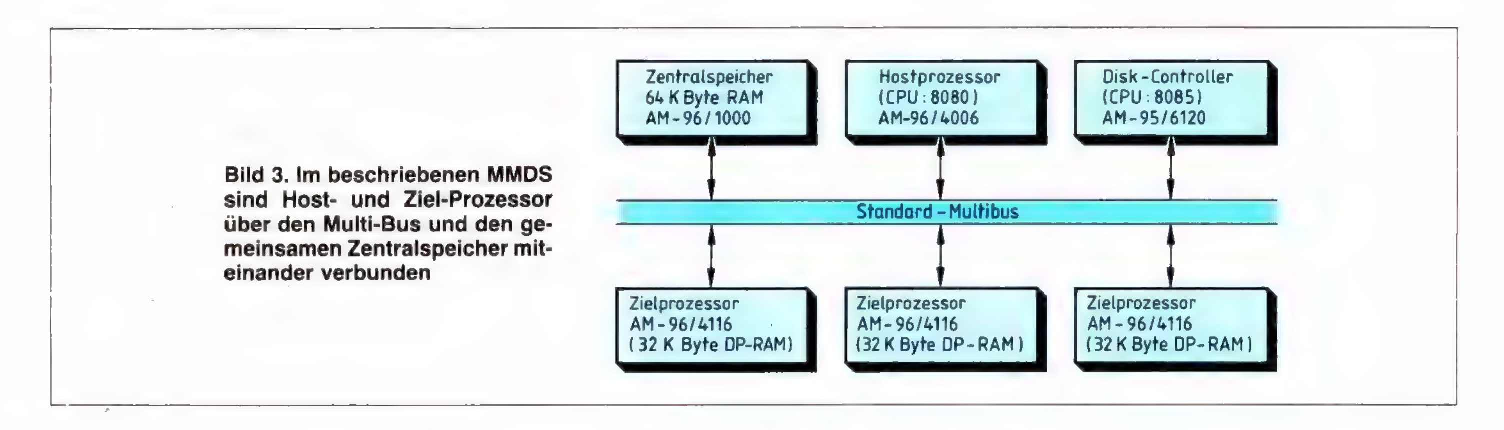


Bild 3. Im beschriebenen MMDS sind Host- und Ziel-Prozessor über den Multi-Bus und den gemeinsamen Zentralspeicher miteinander verbunden

Platine AM-96/4116) vorhanden. Der Host-Prozessor kann nur auf den Zentralspeicher zugreifen, während die Zielprozessoren Zugang zum zentralen und verteilten gemeinsamen Speicherbereich haben.

Um fehlerfreien Betrieb erreichen zu können, müssen folgende Softwareteile zusätzlich vorgesehen werden:

1. System-Steuer-Routinen zur Anpassung der Benutzerschnittstelle an die konkurrierende Umgebung der mehrfach vorhandenen Zielprozessoren.
2. Interface-Routine zwischen dem System-Controller und dem Betriebssystemkern VRTX (Versatile Real Time Executive).
3. Bibliotheks-Routinen zur Unterstützung des Benutzers beim Schreiben von Anwendungsprogrammen für Multiprozessor-Umgebungen.

Das Verständnis der gegenseitigen Beziehungen dieser Programmroutinen mit dem Betriebssystem des Hostcomputers sowie den Benutzer-Tasks und VRTX-Kern der Zielprozessoren ist ebenso wichtig, wie die Kenntnis, in welchem Speicher diese abgelegt sind. Man muß unterscheiden zwischen Nur-Lese- und Schreib-/Lese-Speicherbereichen auf der Host-Prozessor-Karte, dem Zentralspeicher und den N Zielprozessor-Platinen (*Bild* 4).

Bei dem hier vorgeschlagenen MMDS befinden sich die Steuerroutinen des Hostprozessors sowie die Mailboxes für die Kommunikation zwischen den Prozessoren im Zentralspeicher auf der 64-KByte-RAM-Platine. Die Betriebssystem-Software (Monitor für Benutzer-E/A) ist in einem 2-KByte-EPROM untergebracht. Die Benutzer-Task und die Bibliotheks-Routinen (die an die Benutzer-Task angehängt sind) des Zielprozessors befinden sich in den 32-Kbyte-Dual-Port-RAMs, das Betriebssystem (Monitor plus VRTX-Kern) in einem 4-KByte-EPROM. Diese Aufteilung ist notwendig, weil der Host-Prozessor selbst über kein RAM verfügt und nur 64 KByte Speicher adressieren kann.

Der Hostprozessor lädt die Benutzer-Tasks aus dem Plattenspeicher in den Zentralspeicher. Von dort werden sie in die Dual-Port-RAMs der Zielprozessoren transferiert. Der Benutzer kann die Ausführung in jedem Zielprozessor starten und stoppen, indem er die „Create/Delete"-System-Steuer-Routinen benutzt. In ähnlicher Weise kann der Benutzer Tasks zurückstellen, wiederaufnehmen und anfordern, wobei die bewährten Interface-Routinen, die die Steuerung an das VRTX übergeben, zur Anwendung kommen. Mit Systemrufen und der Unterstützung der Bibliotheks-Routinen können Benutzer-Tasks auch VRTX-Routinen aufrufen. Die gesamte Kommunikation innerhalb des Zielprozessors wird über Mailboxes im gemeinsamen Zentralspeicher ausgeführt, kann aber auch über die Dual-Port-RAMs geschehen.

5 Beispiele

Um die gegenseitigen Beziehungen der verschiedenen Softwareeinheiten untereinander zu verdeutlichen, wird im folgenden der Betrieb des vorgeschlagenen MMDS anhand von drei Beispielen dargestellt.

5.1 *Laden einer Benutzer-Task von der Platte in den Speicher des Zielprozessors*

Der Ladeprozeß wird durch das Kommando „LOAD", das man über das Terminal gibt, gestartet. Als Antwort darauf startet der System-Manager SM (der Satz von System-Steuer-Routinen, die auf dem Hostprozessor ausgeführt werden) einen interaktiven Dialog, bei dem der Benutzer folgendes einzugeben hat:

- Dateiname der Task, die geladen werden soll (z. B. TASK-1). Es wird vorausgesetzt, daß die Task als Datei auf der Platte gespeichert ist.

- Nummer des Zielprozessors, auf den die Task geladen werden soll, z. B. Nr. 3.

Nach diesem Dialog beginnt das SM, TASK-1 auf den Zielprozessor 3 zu laden. Diese Prozedur umfaßt sechs Schritte:

1. Das SM lädt TASK-1 von der Platte in einen Puffer (z. B. Puffer 3) innerhalb des gemeinsamen Zentralspeichers (CMM – Central common Memory).
2. Der SM sendet zur Mailbox 3 im CCM, die dem Zielprozessor 3 zugeordnet ist, die folgende Information:
   - die Puffer-Nummer (z. B. 3) im CCM, in der sich TASK-1 befindet
   - die Länge dieses Puffers

– die Startadresse des Speichers im Zielprozessor (DP-RAM, *Bild 4*) in das TASK-1 geladen werden soll
– das LOAS-Kommando, das die eigentliche Transfer-Operation mit folgenden Schritten auslöst:

3. Zielprozessor 3 liest die Information aus seiner Mailbox 3 (Puffer-Nummer, Puffer-Länge, Startadresse).
4. Zielprozessor 3 kopiert TASK-1 aus dem Puffer 3 im CCM in sein DP-RAM.
5. Zielprozessor 3 sendet eine Antwort-Message an die Mailbox 3, um das Ende der Operation anzuzeigen.
6. SM findet die Antwort-Message in der Mailbox 3 und beendet die Lade-Operation.

5.2 Erzeugung einer Benutzertask unter der Steuerung des Haupt-Terminals

Zweck dieser Operation ist die Information des Betriebssystem-Kerns VRTX des Zielprozessors, daß eine Task erzeugt werden soll, d. h. gesteuert werden muß. Die Operation wird durch ein CREATE-Kommando gestartet, das über das Hauptterminal einzugeben ist. Als Antwort darauf startet der SM einen interaktiven Dialog auf dem Haupt-Terminal, indem der Benutzer folgende Angaben machen muß:

1. Die Nummer des Zielprozessors, auf der die Task erzeugt werden soll, z. B. 2
2. die Nummer der Task, z. B. 17.
3. die Priorität der Task, z. B. 12
4. die Startadresse der Task, z. B. B000, d. h. die Adresse des ersten ausführbaren Codes der Task im Zielprozessor-Speicher.

Das CREATE-Kommando sowie die Informationen der Schritte 2 bis 4 werden vom SM zur Mailbox 2 gesendet. Zielprozessor 2 findet das Kommando in Mailbox 2 und ruft VRTX auf, das wiederum die Task 17 mit der Priorität 12 und einer Anfangsadresse B000 erzeugt und eine Message zurücksendet, wenn diese Operation erfolgreich (oder nicht) ausgeführt ist. Diese Message erhält die Mailbox 2, wo sie vom SM gefunden und weiterverarbeitet wird. Der SM informiert darauf den Benutzer durch Anzeige des Ergebnisses der CREATE-Operation auf dem Haupt-Terminal.

5.3 Zurückstellen einer Benutzer-Task, wenn die Mailbox leer ist

Es sei vorausgesetzt, daß Task 17 auf dem Zielprozessor 2 läuft und an einem Punkt angelangt ist, an dem sie Informationen von einer anderen Task benötigt, z. B. Task 9, die auf demselben oder einem anderen Prozessor läuft. Diese Informations-Transfer-Operation wird über eine Mailbox ausgeführt, die sich im CCM befindet. In der Programmiersprache C ist diese Operation mit folgendem Statement beschrieben:

message = pend(mailboxadress, error);

Die Funktion „pend“ findet die gewünschte Mailbox unter „mailboxadress“ und gibt die Information aus

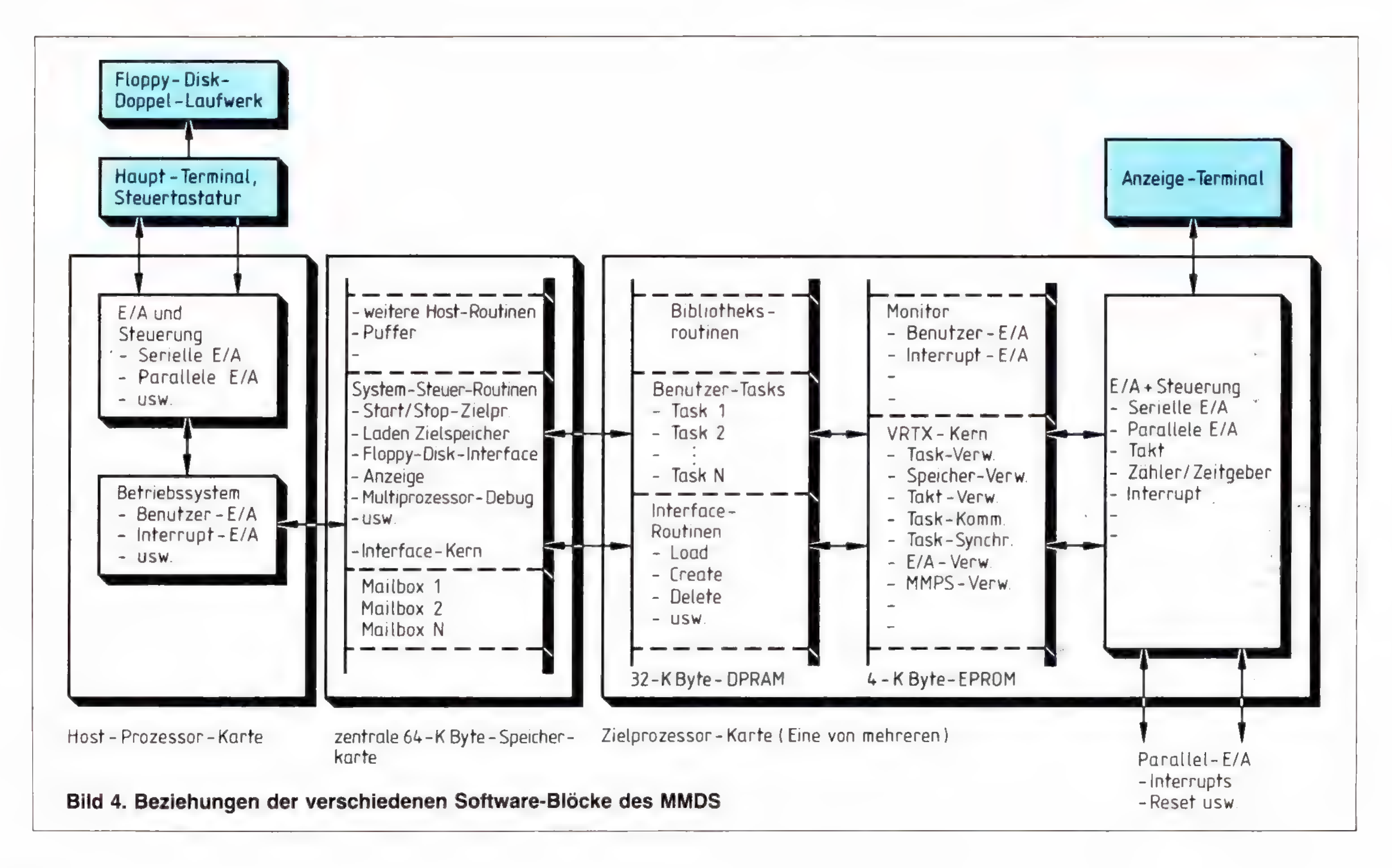


Bild 4. Beziehungen der verschiedenen Software-Blöcke des MMDS

dieser Mailbox in die Variable „message". Task 17 kann sie jetzt benutzen.

Im Fall, daß die Mailbox leer ist, stellt die Funktion „pend" Task 17 zurück. Diese wird von VRTX erneut aufgerufen, wenn die Information vorliegt, daß die erforderliche Message in der Mailbox ist.

6 Erforderliche Steuersoftware

Die Orginal-Software des SMDS besteht aus einem Monitor oder Debugger, die wiederum einfache RS-232-Benutzer-Interface-, Register/Speicher-Inspektions/Modifikations-Routinen usw. umfassen. Beim vorgeschlagenen MMDS müssen die System-Steuerroutinen des Hostcomputers noch weitere Aufgaben erfüllen. Insbesondere muß es die Benutzer-Tasks an den VRTX-Kern anpassen. Aus diesem Grund müssen noch folgende Routinen geschrieben werden:

- Start-/Stop-Routine für den Zielcomputer
  zum Starten und Anhalten der Ausführung von Benutzer-Tasks auf jedem der drei Zielprozessoren durch die Systemsteuerroutinen
- Lade-Routine für den Zentralspeicher
  zum Laden von Benutzer-Tasks von der Platte in den Zentralspeicher, initialisiert durch das Benutzer-Terminal und vom Host-Computer gesteuert
- Datei-Handling-Interface-Routine
  zum Bereitstellen der Unterstützungsfunktionen, die erforderlich sind, um Platten-Files durch Benutzer-Tasks des Zielprozessors öffnen, lesen, beschreiben und schließen zu können
- Display-Fenster-Erzeugungs-Routine
  zum Anzeigen von Zahlen und Texten von Tasks, die auf jedem der drei Zielprozessoren ausgeführt werden, in vorher festgelegten Bereichen des Bildschirms
- Kern-Interface-Routinen
  zum Bereitstellen der erforderlichen Unterstützungsfunktionen für Systemaufrufe der Benutzer-Tasks an das Betriebssystem sowie für die Rückkehr von Systemaufrufen. Folgende Multi-Tasking-Routinen werden unterstützt:
  - Erzeugen, Löschen, Zurückstellen und Wiederaufnehmen von Tasks
  - Initialisierung des Ladevorganges, der die Benutzer-Tasks aus dem gemeinsamen Speicher in ein Dual-Port-RAM bringt
  - Aussenden von Messages usw.
- Multitasking-/Multiprozessor-Debug-Routinen zum Lesen, Anzeigen und Modifizieren von Daten oder Instruktionen in Registern oder Speichern, auf dem System- oder lokalen Bus usw., nachdem das System angehalten oder in den Einzel-Schritt-Betrieb versetzt worden ist.

Die grundsätzliche Philosophie beim Debugging ist die Ausführung von mehreren Tasks auf Multi-Prozessor-Systemen nur, nachdem die eigentliche Task auf einem Einzelprozessor getestet und fehlerfrei gemacht worden ist.

Es ist zu erwarten, daß die System-Steuerroutinen, wie sie hier beschrieben sind, sich verändern werden, wenn die Verfahren und Methoden für Entwicklung, Test und Debugging von Multiprozessor-Software und Hardware für das vorgeschlagene System verändert werden. Existierende Routinen werden verbessert, neue hinzugefügt (besonders, um die Möglichkeiten zum Fehlerbeseitigen zu verbessern), und alle Routinen werden auf mehr gemeinsame Unterprogramme zurückgreifen.

7 Ersatz des Ziel-Computer-Monitors durch VRTX

Ein Platinen-Computer wie z. B. der Typ AMD AM-96/4116 oder Intel SBC-86/12 verfügen über einen einfachen Monitor (System-Steuersoftware) der allgemeine Debug-Funktionen wie Lesen, Anzeige und/oder Modifizieren von Registern oder Speicherplätzen umfaßt. Diese Platinen bieten auf einfache Kommunikation zwischen Prozessoren über zentrale oder verteilte gemeinsame Speicher. Allerdings können diese Programme keine Mehrfach-Tasks verarbeiten oder Kommunikation zwischen Prozessoren synchronisieren. Außerdem muß das gesamte Interrupt-Handling, Echtzeitfunktionen, Speicherzuweisungen und Zeichen-E/A durch Benutzer-Software ausgeführt werden.

Während des letzten Jahres sind verschiedene Betriebssystemkerne auf den Markt gekommen, die als Festkörper-Bausteine verfügbar sind (*Tabelle*). Der

Tabelle der Betriebssysteme, die als Festwertspeicher erhältlich sind

Parameter	80130 (Intel)	ZRTS (Zilog)	VRTX (H & R)
Zielprozessor	8086/186/286	Z8000/1/3/4	8086/Z8000/68000
Umfang/Medium/Erweiterbarkeit	16 K/ROM/Fest	4 K/EPROM/Erw.	4 K/EPROM/Erw.
Kompatibilität zu	RMX-86	nein	nein
Task-Verwaltung	ja	ja	ja
Semaphore	ja	ja	kein Standard
Messages	ja	ja	nein (Mailboxes)
Interrupt-Handling	ja	ja	ja
Speicher-Zuweisung	ja	ja	ja
Erzeugen/Löschen von Tasks	ja	ja	ja
Konfigurations-Methode	Daten-Basis-SW	Konfigurierungs-SW	14 Register

Zweck dieser Kerne ist die Anpassung von Anwendungs-Software in standardisierter Weise (hardware-/software-unabhängig) an Mikroprozessor-Hardware. Dazu wird ein sogenannter Betriebssystem-Bus verwendet. Dem Benutzer bieten sich mit diesen Kernen bewährte und erprobte Routinen für die meisten gebräuchlichen Multitasking-Systemfunktionen.

In dem hier vorgestellten Beispiel wurde VRTX (Versatile Real Time Executive) gewählt, um den Monitor des AM-96/4116 zu ersetzen, insbesondere weil dieses über Mechanismen verfügt, mit denen man den existierenden Vorrat an Kern-Routinen durch spezielle Benutzerroutinen erweitern kann. Obwohl VRTX bereits über eine umfangreiche Menge von Multi-Tasking-Routinen verfügt, unterstützt die vorliegende Version keine Multi-Prozessing-Systemoperation. Deshalb wurde eine Spezial-Routine für die Synchronisation und die Ausführung von Mehrfach-Tasks auf Multi-Prozessor-Systemen zum existierenden Vorrat an Kernroutinen hinzugefügt.

Literatur

[43] *Gladstone, B.:* Arcitecture based on multiple µPs, Buses boost in-circuit emulation. Electronic Design, April 9, 1979, S. 52...55.

[44] 9516 Microsystem Integration Station. Firmenschrift MS-113/8-81/10M der Millenium Inc.

[45] *Schindler, M.:* Can one µC development system be 200 times more powerful than another; Electronic Design, December 25, 1978, S. 60...65.

[46] *Gladstone, B.:* Comparing microcomputer development system capabilities. Computer Design, February 1979, S. 83...90.

[47] *McLeod, J.:* ...while development tools support multiple unsers. Electronic Design, May 10, 1980, S. 141...154.

[48] *Donelly, J. B.:* Emulators für Microprocessor system development. Hewlett-Packard Journal, October 1980, S. 13...20.

[49] Microcomputer Development Systems. Firmenschrift SG54-13706, 1980, Motorola.

[50] 8550 Microcomputer Development Lab, Firmenschrift 61AX-4522, 1980, Tektronix.

[51] Z-LAB 8000, Product description. Firmenschrift der Zilog Inc.

[52] *Braun, J:* Portable development system blazes new trail. Electronics, June 2, 1982, S. 163...168.

[53] *Harbers, H.:* Universal development tool lets designers use best µP. Electronic Design, June 24, 1982, S. 149...156.

Wie stellt man ein preiswertes Multiprozessor-Entwicklungssystem zusammen?

Mit den beschriebenen Modifikationen und wenigen Einplatinen-Computern können die meisten Ein-Prozessor-Entwicklungssysteme auch zur Unterstützung von Multi-Prozessor-Operationen verwendet werden, insbesondere bei solchen mit gemeinsamem Bus im Multiplexbetrieb (TSCB). Die Anzahl der Prozessoren in einem solchen System ist in erster Linie durch die Menge der noch freien Einschubmöglichkeiten im Entwicklungsgeräte-Chassis und durch das Prioritäts-Codierungs-Verfahren beschränkt. Die folgenden Schritte fassen die notwendigen Vorgänge zur Implementierung des MMDS zusammen:

● Kauf folgender Gegenstände:
- Konventionelles µP-Entwicklungssystem mit Multi-Master-Bus
- sowie MDS-Software-Unterstützung und Hochsprachen-Compiler
- N Bus-kompatible Einplatinen-Computer der µP-Familie des Zielrechners
- Betriebssystemkern im ROM, z. B. VRTX

● Initialisierung und Inbetriebnahme des Systems
- Zusammenbau von Host-Computer, gemeinsamem Speicher und E/A-Platinen
- Zusammenbau der Ziel-Computer (z. B. Festlegen des Adreßraumes durch Schalter oder Drahtbrücken)
- Einschub der Zielcomputer in die freien Plätze des SMDS
- Ausführung einzelner Testroutinen auf jedem Prozessor

● Schreiben der System-Steuerroutinen für den Hostrechner zum
- Laden von Anwendungsprogrammen in den Zielprozessor
- Aufnahme und Anzeige von Daten aus Mailboxes im gemeinsamen Speicher
- Rücksetzen und Synchronisieren der Frame-Rate des Zielprozessors
- Anhalten bei einem Fehler und Anzeigen der kritischen Daten usw.

● Hinzufügen eines Multi-Tasking-Kerns, so daß
- jeder Prozessor mehrere Tasks erzeugen, aussetzen und ablaufen lassen kann
- Speicher, E/A und andere gemeinsame Ressourcen verwalten kann
- Echtzeit-Takt- und Interrupt-Funktionen steuern kann

● Hinzufügen von Multiprozessor-Funktionen zum Multi-Tasking-Kern
- Synchronisierung der Kommunikation zwischen den Prozessoren
- Umsetzen logischer in physikalische Adressen

● Überprüfen der Funktion mit Beispielprogramm
- Ausführung des Demonstrationsprogramms auf einem Prozessor
- Ausführung des Demonstrationsprogramms auf mehreren Prozessoren

● Entwicklung von Anwendungs-Software für jeden Prozessor unter Verwendung von
- Standard-Entwicklungssystemen-Unterstützungs-Software
- Konventionellen Editoren, Assemblern, Compilern und Bindern
- Assemblierung und HLL-Programmierung
- einfachen und mehreren Tasks mit Betriebssystem.

Dr. Egon Hörbst, Dipl.-Ing. Anton Sauer

Ein Multi-Mikrocomputer-System am Arbeitsplatz

1. Teil: Konzept und Zielsetzung

Der Einsatz von Multi-Mikrocomputer-Systemen ist in dem weiten Bereich von der intelligenten Schreibmaschine bis zur kommerziellen oder industriellen Datenverarbeitung möglich und sinnvoll – in Zukunft auch in Großrechnersystemen. Die Vorteile dieser Strukturen sind – nach dem großen Einsatzspektrum – die funktionelle Aufteilung von Hardware und Software, die Modularität des Systemaufbaues und die dadurch bedingte Erweiterbarkeit. Die Aufgaben reichen von der Textbearbeitung im Schreibbüro bis zur Systementwicklung in Labor und Prüffeld. Im 1. Teil dieser fünfteiligen Aufsatzreihe, die von einem siebenköpfigen Autoren-Team aus den Forschungslaboratorien der Siemens AG verfaßt wurde, werden nach der grundsätzlichen Vorstellung von Hard- und Software eines Mehrrechnersystems am Arbeitsplatz die Vorteile bei der Systementwicklung, beim Kunden und bei der Systembetreuung durch Hersteller und Verteiler diskutiert.

1 Einsatzgebiet von Multi-Mikrocomputer-Systemen und Anforderungen

Mehr-Computersysteme auf der Basis von Mikroprozessoren können überall dort eingesetzt werden, wo eine Aufteilung der Software auf selbständige Hardwarekomponenten (Mikrocomputer) zulässig ist. Der funktionelle Ablauf kann dabei zentral gesteuert erfolgen oder von einer Systemkomponente übernommen werden [1].

Mögliche Ausführungen solcher Strukturen, bestimmte Lösungsvorschläge und Anordnungsvorschläge sind in [2, 3, 4] enthalten, wobei sich die Aussagen bezüglich der Brauchbarkeit der Struktur auf bestimmte ausgewählte Beispiele beschränken.

Beispiele sind Einsatzmöglichkeiten

- in Informationssystemen (Management, Leitzentralen),
- im gesamten Gebiet der mittleren Datentechnik (Arbeitsplatzcomputer, Bürocomputer),
- in künftigen Großrechnersystemen [5].

Diese Art der Informationsverarbeitung ist auch von Monoprozessorsystemen her bekannt. Mehrrechnersysteme bieten jedoch einige Vorteile. Sie liegen:

- in der Modularität des Systemaufbaues und der dadurch bedingten Erweiterbarkeit,
- in der funktionellen Aufteilung von Hard- und Software
- im großen Einsatzspektrum.

Daraus folgt für Mehrrechnersysteme, daß die Systementwicklung gezwungenermaßen strukturiert erfolgen muß und daß der Test der Einzelkomponenten weitgehend unabhängig erfolgen kann.

Für ein Rechnersystem, das am Arbeitsplatz eingesetzt wird, ergeben sich Anforderungen, die vom Anwender (Sachbearbeiter, Sekretärin) gestellt werden und aus den Aufgaben resultieren, die mit einem solchen Arbeitsplatzcomputer gelöst werden sollen.

Beispiele dafür sind:

- Textbe- und -verarbeitung
- Daten speichern, ändern, suchen, sortieren, aufbereiten, abrufen, anzeigen
- Kommunikation mit anderen Systemen (Anschluß an Datennetze, an andere Rechner, Verbindung solcher Systeme untereinander)
- Entwicklung von Systemen (Hardware, Software von Rechnern, Anwendersoftware, grafische Datenverarbeitungssysteme, usw.)
- Test von Systemen.

Die Thematik der Aufsatzreihe ist folgendermaßen unterteilt.

1. Teil: Konzept und Zielsetzung
2. Teil: Bus-Strukturen für Mehrrechnersysteme
3. Teil: Verteiltes Betriebssystem einer Mehrrechnerstruktur
4. Teil: Praktische Ersatzbeispiele eines Mehrrechnersystems als Terminal
5. Teil: Entwicklung, Test und Aufbau von Mehrrechnersystemen

Die Art dieser Beispiele zeigt, daß ein derartiger Arbeitsplatzcomputer in einem Schreibbüro, Sekretariat, in Vertriebs-, Fertigungs- oder Entwicklungsabteilungen eingesetzt werden kann (siehe Teil 4). Ein Vorteil des Einsatzes einer Mehrrechnerstruktur in solch unterschiedlichen Umgebungen liegt eben darin, daß nur ein einziges Grundsystem zu entwickeln und herzustellen ist, das in unterschiedlichen Ausbaustufen mit unterschiedlichen Softwarepaketen für diesen weiten Bereich geeignet ist.

2 Konzeption eines Multi-Mikrocomputer-Systems

Um die genannten Anforderungen realisieren zu können, ist es notwendig, eine Aufteilung des Gesamtsystems (Software und Hardware) in einzelne Funktionen vorzunehmen. Eine Aufteilung in Funktionen und damit die Übertragung der Funktionen auf eine Mehrrechnerstruktur wird aus einem Monorechnersystem abgeleitet. In der *Tabelle* sind einige Funktionen ausgewählt und deren Aufgaben angegeben.

Aus der funktionellen Aufteilung läßt sich eine Einteilung der in der Tabelle genannten Funktionen in drei Ebenen vornehmen:

- Dialogebene mit der Dialogfunktion,
- Kommunikationsebene mit der internen Kommunikationsfunktion und
- Funktionsebene mit den restlichen Funktionen nach Tabelle (Verarbeitungs-, Ein/Ausgabe-, Dateibearbeitungs-, Diagnose-, Test-, externe Kommunikationsfunktion).

Zusätzlich kommen noch Benutzer- und Peripherieebene hinzu.

Über Benutzer- und Dialogebene wird zu gegebener Zeit eine bestimmte Funktion der Funktionsebene ausgewählt und über die Kommunikationsebene erreicht. Dort wird die Abarbeitung dieser Funktion initialisiert. Während der Abarbeitung dieser Funktion sind Kommunikations-, Dialog- und Benutzerebene wieder freigegeben für andere Abläufe.

Ein weiterer Kernpunkt bei der Konzipierung eines Mehrrechnersystems ist die Kommunikation der einzelnen Bestandteile des Systems untereinander. Einige Kommunikationsmechanismen sind in Theorie und Praxis für Multiprozessorsysteme untersucht und beschrieben worden [6, 7, 8, 9]. Darüber hinaus wurde in [10] eine graphentheoretische Klassifikation nach dem Typ der Verbindung zwischen den Knoten vorgestellt und in [11] eine Einteilung nach der Struktur des Arbeitsspeichersystems vorgenommen.

Zur Realisierung der Anforderungen kann man auf streng hierarchisch aufgebaute Systeme, wie sie heute in der Prozeßverarbeitung noch eingesetzt werden, verzichten. Die Hardwarekomponenten von Dialog- und Funktionsebene werden dezentral, gleichberechtigt um die Kommunikationsebene angeordnet. Mögliche Architekturen sind dann Ringstruktur, Sternstruktur und vermaschte Struktur [10].

Untersuchungen an einer vermaschten Struktur und einer Sternstruktur in einem dezentralen Mehrrechnersystem haben ergeben, daß die Sternstruktur der vermaschten hinsichtlich Erweiterbarkeit und Flexibilität weit überlegen ist [12]. Hinsichtlich Sicherheit dürfte die Sternstruktur (bei Doppelung) der Ringstruktur überlegen sein. In unserem Fall wurde deshalb eine Sternstruktur realisiert (siehe auch Teil 2).

Tabelle. Auswahl von Funktionen und deren Aufgaben für den Arbeitsplatzcomputer

Funktion	Aufgaben
Dialogfunktion	– Maskenaufbereitung Benutzerführung Eingabeprüfung
Verarbeitungsfunktion	– Übersetzen, Binden,... mathem. Funktionen Textedition
Ein/Ausgabefunktion für periphere Geräte	– Kanalroutinen Textaufbereitung
Dateibearbeitungsfunktion	– Verwaltung Suchen Sortieren
Diagnose- und Testfunktion	– Systemtest Erkennung von Hw- und Sw-Fehlern Maßnahmen zur Fehlerbehebung
interne Kommunikationsfunktion	– interner Datenaustausch Verhinderung von Systemzusammenbrüchen
externe Kommunikationsfunktion	– Anschluß an andere Rechner Prozedurabwicklung Textaufbereitung

3 Systemaufbau

3.1 Hardwareaufbau

Die in der Tabelle aufgezählten Funktionen können nun auf unterschiedliche Weise auf eine Mehrrechnerstruktur übertragen werden [13]. Gewählt wurde eine Aufteilung in verschiedene Ebenen, wie sie in [10] beschrieben ist. Im einfachsten Fall wird jede Funktion auf einem eigenen Rechner installiert, so daß sich, wenn man Redundanzen berücksichtigt, die Struktur in *Bild 1* ergibt.

Die Hardwarekomponenten des Systems (Mikrocomputer) sind modular aufgebaut. Jeder Rechner verfügt über ein gleiches Grundmodul. Daneben können, abhängig von der Funktion, die dem Rechner zugeordnet ist, Speichermodule und E/A-Module für spezielle periphere Anschaltungen dazu gesteckt werden. Das Grundmodul besteht aus der CPU-Gruppe, Programm- und Datenspeicher, paralleler und se-

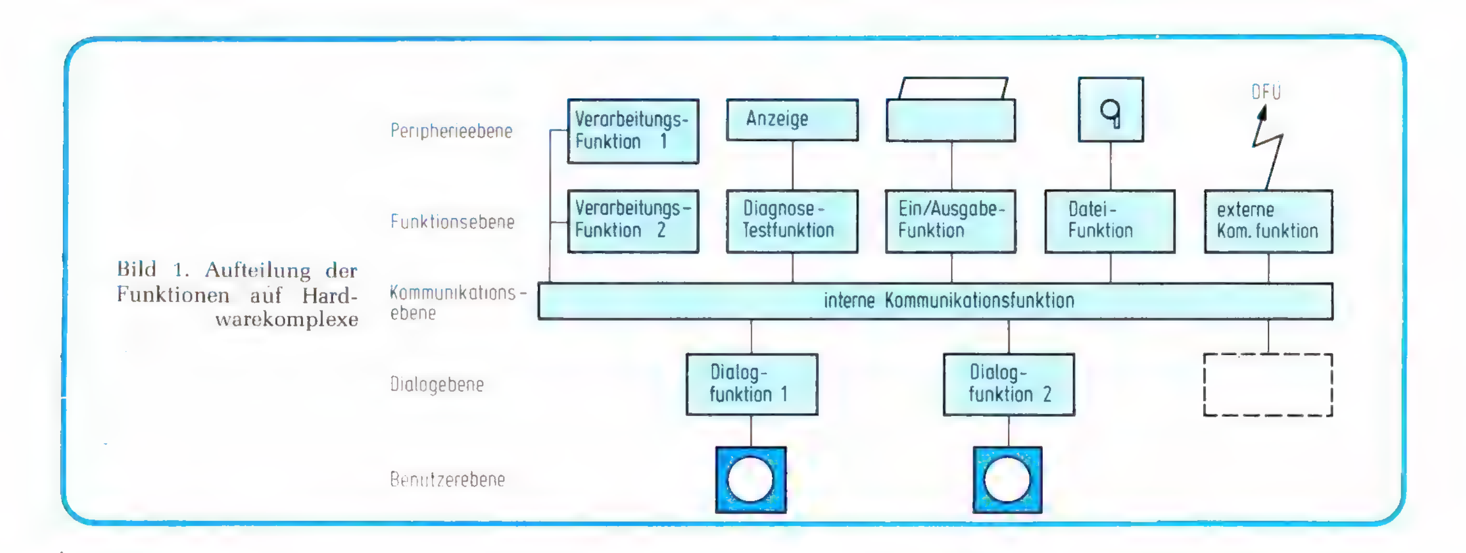


Bild 1. Aufteilung der Funktionen auf Hardwarekomplexe

rieller Standardschnittstelle und einem Interface zur Kommunikation zwischen den einzelnen Rechnern.

Der Datenaustausch zwischen den Rechnern erfolgt über einen gemeinsamen Bus. Es wurde ein hardwaregesteuerter Datenaustausch über direkten Speicherzugriff (DMA) realisiert [14]. Auf diese Weise sind mit heute erhältlichen DMA-Controllern Übertragungsraten bis zu 1 MByte/s erreichbar.

Der Start des Gesamtsystems mit den einzelnen Komponenten läuft vollautomatisch ab. Das Diagnose- und Testsystem liefert am Anzeigefeld eine Aussage über die Betriebsbereitschaft des Systems. Eine weitere Aufgabe des Diagnose- und Testsystems ist die Überwachung auf Ausfälle von Einzelkomponenten, wobei Fehlermeldungen auf dem Anzeigenfeld angezeigt werden.

3.2 *Softwareaufbau*

Die funktionale Aufteilung des Systems auf einzelne Hardwarekomponenten (Mikrocomputer) bedeutet eine Zerlegung der Systemsoftware in funktionell unabhängige Teile, die in ein hierarchisch strukturiertes Niveauschema eingebaut werden [15] (siehe Teil 3).

Ein solches Niveauschema am Beispiel des Arbeitsplatzcomputers nach Bild 1 ist in *Bild 2* wiedergegeben.

Der Aufbau des Systems beginnt auf der untersten Ebene (Hardware-Ebene) mit der lokalen Software, z. B. Gerätetreiber.

Über der lokalen Software befindet sich die Kommunikationssoftware. Die Kommunikationsprozeduren sind auf allen Rechnern im wesentlichen gleich. Damit wird vor allem die Modularität und Erweiterbarkeit des Gesamtsystems gewährleistet. Die Kommunikationssoftware erfüllt die Aufgaben der internen Datenübertragung und der Rechnersynchronisation.

Das Betriebssystem umfaßt Serviceroutinen für Datentransport und Programmausführung. Der Zugang von der Anwenderebene zur Betriebssystemebene erfolgt über die Systemschnittstelle.

Mit diesem Hierarchiekonzept und der Unabhängigkeit der Programme innerhalb einer Ebene wird eine hohe Sicherheit und Flexibilität auch der Systemsoftware erreicht.

Die Anpassung an unterschiedliche Einsatzfälle und Ausbaustufen ist gewährleistet und wird während der Systemgenerierung durchgeführt. Dies kann durch einen Lade-ROM und durch Einlesen des Betriebssystems von einem externen Speicher, z. B. Diskette erfolgen. Bei Änderungen in der Konfiguration ist dann der entsprechende Datenträger zu ändern.

4 Zielsetzung

Drei Hauptziele waren vorgegeben für die Entwicklung eines Arbeitsplatzcomputers auf der Basis eines Mehrrechnersystems. Dieses System soll gegenüber herkömmlichen Systemen

- Vorteile bei der Systementwicklung bringen,
- die Flexibilität des Systemeinsatzes beim Kunden gewährleisten und
- die Wirtschaftlichkeit bei der Systembetreuung durch den Hersteller und den Vertreiber erhöhen.

4.1 *Vorteile bei der Systementwicklung*

Durch die funktionelle Aufteilung der Software auf gleichartige Hardwaregebilde wird ein modularer Systemaufbau gewährleistet. Auf jedem Rechner wird ein genau abgegrenzter Softwareteil implementiert. Für die Systementwicklung bedeutet dies, daß zwi-

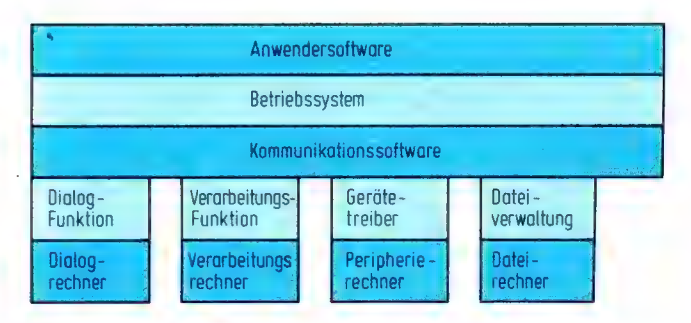


Bild 2. Struktur der Systemsoftware des Arbeitsplatzcomputers nach Bild 1

schen den einzelnen Funktionen nicht nur klare Hardware-Schnittstellen, sondern auch Software-Schnittstellen vorhanden sind.

Ein weiterer Punkt ist die Testsituation bei der Systementwicklung. Es muß immer nur die einzelne Funktion, das einzelne Modul, das einzelne Gerät getestet werden.

4.2 *Flexibilität beim Systemeinsatz*

Geht man davon aus, daß unser Mehrrechnersystem mit den aktuellen Anforderungen mitzuwachsen hat, dann heißt das, daß das System in seiner ursprünglich gewünschten Ausbaustufe nicht für immer eingesetzt wird. Ein wichtiger Punkt ist daher die Systemerweiterbarkeit und zwar um mehr oder weniger Funktionen. Voraussetzung dafür ist die Ladbarkeit der Programme in die einzelnen Rechner.

Das System ist „vor Ort" generierbar. Je nach Ausbau an Rechnern, Funktionen, Peripherie wird das aktuelle Betriebssystem am Einsatzort generiert. Beim Einschalten und Hochfahren des Systems genügt dann in jedem Rechner ein Urlade-ROM, die aktuelle Systemsoftware wird vom externen Speicher in den Schreib-Lese-Speicher der einzelnen Rechner geladen.

4.3 *Wirtschaftlichkeit bei der Systembetreuung*

Der zuletzt genannte Punkt, die Generierbarkeit des Systems beim Kunden bringt dem Hersteller und Kunden einen großen Vorteil: Die Lagerhaltung wird stark vereinfacht, weil sie sich auf die Urlade-ROMs und auf ein allgemeines Betriebssystem beschränkt und nicht auf Software, die auf den Systemausbau bei jedem Kunden zugeschnitten ist.



Dr. phil. Egon Hörbst (links) ist gebürtiger Innsbrucker. Er studierte in Innsbruck Mathematik und Astronomie und trat 1970 in die Siemens AG ein. Seit dieser Zeit beschäftigte er sich mit Systemen für grafische Datenverarbeitung, Prozeßrechnern und Mikroprozessoren.
Hobbys: Skifahren, Wassersport.
Telefon: (0 89) 67 82-33 54.
ELEKTRONIK-Leser seit 1970.

Dipl.-Ing. Anton Sauer (rechts), von Geburt Augsburger, studierte an der Technischen Hochschule München Nachrichtentechnik und trat 1969 in die Siemens AG ein. Nach Arbeiten auf dem Gebiet der Datenübertragungstechnik beschäftigt er sich seit 1975 mit Mikrocomputer-Systemen.
Hobbys: Bergwandern, Fotografieren.
Telefon: (0 89) 67 82-42 55.
ELEKTRONIK-Leser seit 1962.

Ein ganz anderer Aspekt ist die Vereinfachung der Wartung. Da das System modular aus gleichen Einheiten aufgebaut ist, wird auch die Wartbarkeit sehr vereinfacht. Der Wartungstechniker wechselt bei fehlerhaftem Verhalten des Systems einzelne gleichartige Platinen aus und kann so den Fehler auf die einzelne Baugruppe einschränken. (Die Fehlersuche auf der fehlerhaften Baugruppe wird nicht beim Kunden, sondern beim Hersteller vorgenommen.) Die Fehlereingrenzung wird – wiederum bedingt durch den modularen Systemaufbau – in verschiedenen Stufen vorgenommen: Test des Gesamtsystems, des Kommunikationsteils, des fehlerhaften Rechners (seiner CPU, seines Speichers), des fehlerhaften Endgerätes.

Ein wiederum anderer Aspekt ist das einfache Akquirieren beim Kunden mit auf die aktuellen Probleme des Kunden zugeschnittener Anfangskonfiguration. Durch die Breite des Einsatzspektrums unserer Mehrrechnerstruktur für sehr geringe Anforderungen (z. B. intelligente Schreibmaschine) bis zu sehr hohen Anforderungen (z. B. Beginn der kommerziellen Datenverarbeitung) mit ein und derselben Hard- und Software (in unterschiedlicher Anordnung) ist es möglich, die meisten Probleme des Kunden auf dem Gebiet der mittleren Datentechnik zu lösen.

Literatur

1 *Sauer, A.:* Sequential System Structures. Siemens Forschungs- und Entwicklungsberichte 7 (1978), H. 6, S. 319...321.

2 *Burton, T.:* Multi-μP-systems combine the efficiency of dedicated microcomputers with the throughput of minis. Electronic Design 25 (1977), H. 16, S. 68...71.

3 *Reyling, G.:* Performance and Control of Multiple Microprocessor Systems. Computer Design 13 (1974), H. 3, S. 81...86.

4 *Colon, F. et al.:* Coupling small computers for performance enhancement. Proceedings of the National Computer Conference 1976, S. 755...764.

5 *Donner, H.:* Tendenzen für die Architektur von Rechensystemen. NTG-Fachberichte, Band 62, 1978, S. 21...38.

6 *Anderson, G. A.; Jensen, E. D.:* Computer Interconnection Structures: Taxonomy, Characteristics, and Examples. Computing Surveys 7 (1975), H. 4, S. 197...213.

7 *Weissberger, A. J.:* Analysis of Multiple-Microprocessor System Architectures. Computer Design 16 (1977), H. 6, S. 151...163.

8 *Baer, J.-L.:* Multiprocessing Systems. IEEE Transactions on Computers C-25 (1976), H. 12, S. 1271...1277.

9 *Spetz, W. L.:* Microprocessor Networks. Computer 10 (1977), H. 7, S. 64..70.

10 *Hörbst, E.; Sauer, A.; Weiss, J.:* Verkuppelt. Klassifikation der Strukturen von Mikrocomputersystemen. Micomp 2 (1977), H. 6, S. 40, 42, 44 und 3 (1978), H. 1, S. 40, 43, 44.

11 *Witte, J.:* Lokale Speicher in Multimikroprozessorsystemen. Micomp 3 (1978), H. 8, S. 24, 25 und 3 (1978), H. 9, S. 24, 25, 27.

12 *Bellm, J.; Hörbst, E.; Sauer, A.:* Vergleich einer vermaschten Struktur mit einer Sternstruktur in einem dezentralen Mehrrechnersystem. International Microcomputers, Minicomputers, Microprocessors, Geneva, 1977, S. 115...122.

13 *Tibbals, H. F.; Curran, P.:* Optimizing function distribution in a terminal network. Microprocessors 1 (1977), H. 6, S. 362...368.

14 *Bellm, H.; Sauer, A.:* Methods of data exchange between microcomputers. 3rd Euromicro Symposium, Amsterdam, October 1977 (Preprints), S. 16...22.

15 *Kroneberg, A.:* Hierarchisch strukturierte System-Software. Siemens Forschungs- und Entwicklungsberichte. 7 (1978), H. 6, S. 348...351.

16 *Singer, W.; Attweger, W.; Patzelt, R.:* Koordination eines Multiprozessorbusses durch Stichleitungen. ELEKTRONIK 1978, H. 10, S. 81...83.

17 *Engelhardt, H.; Feger, O.:* Multi-Mikrocomputersysteme. ELEKTRONIK 1978, H. 11, S. 49...53.

18 *Raphael, H.:* Mehrfachverarbeitung mit Multiprozessorsystemen. ELEKTRONIK 1978, H. 11, S. 84...87.

Dipl.-Inform. Hans Bellm, Ing. (grad.) Hans Thinschmidt

Ein Multi-Mikrocomputer-System am Arbeitsplatz

2. Teil: Bus-Strukturen für Mehrrechnersysteme

Multi-Mikrocomputer-Systeme sind in sehr vielen Anwendungsbereichen einsetzbar und erfreuen sich deshalb bei der Systementwicklung wachsender Beliebtheit. Um in die verwirrende Vielfalt der Architektur-Möglichkeiten und Begriffsbestimmungen etwas Klarheit zu bringen, werden zunächst einige Beispiele zur Klassifizierung solcher Systeme angegeben.

Ein Kernpunkt bei der Konzipierung eines Mehrrechnersystems ist die Art des Datenaustausches der einzelnen Rechner untereinander. Verschiedene Übertragungsmechanismen werden angegeben, die sich in der Verwendung verschieden komplexer Hardware-Hilfsmittel unterscheiden. Grundsätzlich kann festgestellt werden, daß unter Zuhilfenahme von immer leistungsfähigerer Hardware der Software-Anteil bei den Übertragungsprozeduren verringert und damit die Übertragungsrate erhöht werden kann.

Die Architektur eines Mehrrechnersystems wird wesentlich durch das Kommunikationssystem bestimmt, das die einzelnen Rechner untereinander verbindet. Dabei wird insbesondere auf Bus-Systeme eingegangen. Ein Lösungsvorschlag für den Aufbau eines Bus-Systems über direkten Speicherzugriff wird vorgestellt und ausführlich beschrieben.

1 Allgemeine Übersicht und Abgrenzung

Computer kann man vereinfacht als Systeme betrachten, die Befehlsströme (Programme) ausführen, um Datenströme zu verarbeiten. Entsprechend der Anzahl der möglichen Befehls- und Datenströme kann man Computersysteme in vier Klassen unterteilen mit den Merkmalen [1, 2, 3]:

- ein Befehlsstrom und ein Datenstrom (SISD = *single instruction single data*)
- ein Befehlsstrom und mehrere Datenströme (SIMD = *single instruction multiple data*)
- mehrere Befehlsströme und ein Datenstrom (MISD = *multiple instruction single data*)
- mehrere Befehlsströme und mehrere Datenströme (MIMD = *multiple instruction multiple data*).

Die SISD-Klasse beinhaltet die Monoprozessorsysteme, während es sich bei den anderen drei Klassen um Multiprozessorsysteme im weitesten Sinne handelt. Bei Systemen mit SIMD- und MSID-Architekturen handelt es sich um nur wenig verbreitete Spezialrechner, die hier nicht weiter betrachtet werden.

Systeme der MIMD-Kategorie bestehen allgemein aus einer Menge von Systemelementen, die über ein Kommunikationssystem miteinander verbunden sind. Zur weiteren Klassifizierung der MIMD-Kategorie können folgende Charakteristika herangezogen werden [4, 5, 6]:

- Art der Kopplung zwischen den Systemelementen (lose, fest)

Entsprechend der Mächtigkeit der Systeme, die miteinander verbunden sind, unterscheidet man zwischen lose und fest gekoppelten Systemen.
Bei lose gekoppelten Systemen bestehen die einzelnen Systemelemente aus kompletten Computern (Multicomputersysteme), während in fest gekoppelten Systemen bestimmte Systemelemente wie Speicher und Ein-/Ausgabebausteine zur gemeinsamen Nutzung von mehreren Prozessoren zur Verfügung stehen (Multiprozessorsysteme).

- Entfernung zwischen den Systemelementen
  Bei weit entfernten Systemelementen (Rechnern) erfolgt die Kommunikation zwischen den Rechnern über Datenfernübertragungseinrichtungen. Hierzu gehören die Computernetzwerke. Bei räumlich nah angeordneten Mehrrechnersystemen bieten sich mehr Kommunikationsmöglichkeiten und sind engere Verflechtungen in den Problemlösungen möglich.
- Art der Funktionszuordnung (statisch, dynamisch)
  Bei Systemen mit statischer Funktionszuordnung besteht eine feste Bindung der einzelnen Funktionen zu den einzelnen Rechnern, d. h. jeder ist für die Abarbeitung ganz bestimmter Funktionen zu-

ständig. Die Funktionsverteilung wird bei der Systemimplementierung festgelegt.
Bei Systemen mit dynamischer Funktionszuordnung können die einzelnen Funktionen von mehreren oder allen Rechnern ausgeführt werden. Die Auswahl eines geeigneten Rechners, d. h. die Zuordnung Funktion – Rechner erfolgt dynamisch durch das Betriebssystem. Einige Systeme benutzen in Abhängigkeit vom Funktionstyp eine Mischung aus statischer u. dynamischer Funktionszuordnung.

- Priorität der Rechner untereinander (hierarchisch, gleichberechtigt)
  Bei hierarchisch aufgebauten Systemen werden die Rechner niedriger Priorität von denen höherer Priorität gesteuert. Hierzu gehören die Master-Slave-Systeme (2stufige Hierarchie). Die Priorisierung der Rechner kann durch die Architektur vorgegeben sein oder per Software vorgenommen werden. Bei gleichberechtigten Systemen gibt es keine ausgezeichneten Rechner, d. h. alle Rechner sind mit den gleichen Möglichkeiten ausgestattet.
- Architektur des Kommunikationssystems
  Das Kommunikationssystem verbindet die einzelnen Systemelemente miteinander und ermöglicht den Datenaustausch zwischen den einzelnen Rechnern. Hierbei kann man zwischen Systemen mit kollektiven Kommunikationswegen (Bus-Systeme) und Systemen mit individuellen Kommunikationswegen (Ring, Maschennetz, Baum) unterscheiden. In Abschnitt 3 wird dieser Punkt näher behandelt.
- Art des Datenaustausches
  Der Datenaustausch zwischen den Rechnern kann grundsätzlich seriell oder parallel erfolgen. Die Wahl der Übertragungsart ist abhängig von den Geschwindigkeitsanforderungen und dem Verdrahtungsaufwand. Im nächsten Abschnitt werden verschiedene Möglichkeiten zur Implementierung eines parallelen Datenaustausches aufgezeigt. Auf serielle Übertragungsmechanismen wird nicht näher eingegangen.

In den nachfolgenden Ausführungen wird immer wieder Bezug genommen auf das im 1. Teil vorgestellte Multi-Mikrocomputersystem (*Bild 1*), den sogenannten Arbeitsplatzcomputer [7]. Dabei wird zwischen zwei Rechnertypen unterschieden:

– Dialogrechner
  An jeden Dialogrechner ist eine Konsole angeschlossen. Damit besteht eine eindeutige Zuordnung zwischen Benutzer und Dialogrechner, d. h. Dialogrechner arbeiten benutzerorientiert.
– Funktionsrechner
  Auf den Funktionsrechnern sind bestimmte Funktionen implementiert, die von jedem Dialogrechner in Anspruch genommen werden können, d. h. Funktionsrechner arbeiten funktionsorientiert.

Weitere Merkmale des Arbeitsplatzcomputers sind:

– lose gekoppeltes System mit autonomen Rechnern
– räumlich nah angeordnete Rechner
– je nach Funktionstyp statische oder dynamische Funktionszuordnung
– gleiche Priorität bei Rechnern gleichen Typs
– ein Bus-System für die Kommunikation
– paralleler Datenaustausch.

D = Dialogrechner F = Funktionsrechner

Bild 1. Struktur des Arbeitsplatzcomputers

2 Datenaustausch-Verfahren

Die Verfahren für den parallelen Datenaustausch zwischen autonomen Rechnern eines Mehrrechnersystems lassen sich in zwei Klassen einteilen:

– Softwaregesteuerter Datenaustausch
  Dabei wird der Datenaustausch per Software durchgeführt, evtl. unter Zuhilfenahme von Hardware-Hilfsmitteln.
– Hardwaregesteuerter Datenaustausch
  Dabei wird der Datenaustausch per Hardware ohne Beteiligung von Software durchgeführt.

Die nachfolgend angegebenen drei Verfahren sind typische Vertreter der softwaregesteuerten Datenaustausch-Mechanismen. Sie arbeiten jeweils nach dem Handshaking-Prinzip [8].

- Datenaustausch mit Software-Synchronisierung

  Hierbei erfolgt der Datenaustausch rein programmgesteuert ohne Zuhilfenahme von Hardware. Der Datenaustausch zwischen den Rechnern wird per Software über Ein-/Ausgabeports vorgenommen. Die zur Synchronisierung von Sender und Empfänger erforderlichen Steuersignale werden bei diesem Verfahren ebenfalls softwaremäßig generiert und über Ports einander zugesendet. Die hiermit erzielbaren Übertragungsgeschwindigkeiten sind relativ klein.
- Datenaustausch mit Hardware-Synchronisation

  Wie zuvor erfolgt der Datenaustausch zwischen den Rechnern per Software über Ein-/Ausgabeports, die zur Synchronisation von Sender und Empfänger erforderlichen Steuersignale werden jedoch hardwaremäßig generiert. Damit reduziert sich der Software-Anteil an der Übertragungsprozedur (interruptgesteuert) und erhöht sich die Übertragungsrate.
- Datenaustausch über direkten Speicherzugriff

  Hierbei wird der Datenaustausch nicht über Ein-/Ausgabeports, sondern über Software-Zugriffe auf

den Speicher von Sende- und Empfangsrechner vorgenommen. Eine Hardwaresteuerung gestattet beispielsweise dem Senderechner, während des Datentransfers Schreiboperationen auf dem Speicher des Empfangsrechners durchzuführen. Damit ist für den Datenaustausch nur noch in einem Rechner Software erforderlich. Das ermöglicht die höchste Übertragungsrate unter den softwaregesteuerten Übertragungsmethoden.

Bei Datenaustausch-Verfahren kann grundsätzlich die Übertragungsgeschwindigkeit gesteigert werden, wenn für die Realisierung der Übertragungsprozeduren der Software-Anteil reduziert und entsprechende Hardware eingesetzt wird.

Beim hardwaregesteuerten Datenaustausch läuft der Datentransfer ohne CPU-Beteiligung, d. h. ohne jegliche Software ab. Damit hängt die Übertragungsrate auch nicht mehr von der Arbeitsgeschwindigkeit der Rechner ab. Ein typisches Verfahren dieser Klasse ist der hardwaregesteuerte direkte Speicherzugriff (DMA = *direct memory access*).

Die DMA-Steuerung verschafft sich für die Datenübertragung Zugriff auf die Speicherbereiche von Sende- und Empfangsrechner und steuert, ebenfalls im Handshaking-Prinzip, den Datenaustausch. Die erreichbare Übertragungsgeschwindigkeit hängt von der Arbeitsgeschwindigkeit der DMA-Steuerung und den Speicherzugriffszeiten ab und liegt in der Regel um Größenordnungen höher als beim softwaregesteuerten Datenaustausch.

3 Architektur von Mehrrechnersystemen

3.1 Kommunikationssystem

Die Wahl des Datenaustausch-Verfahrens ist relativ unabhängig von dem darauf aufbauenden Kommunikationssystem, das die einzelnen Rechner miteinander verbindet und maßgebend die Architektur des Mehrrechnersystems bestimmt. Beim Entwurf des Kommunikationssystems sind prinzipiell folgende Zielsetzungen anzustreben:

- flexibler Aufbau
- gute Erweiterbarkeit
- optimale Leistung
- hohe Verfügbarkeit
- gute Betriebssicherheit
- hoher Durchsatz.

Diese grundsätzlichen Zielvorstellungen sind entsprechend der vorgesehenen Einsatzgebiete zu gewichten und mit den technischen und ökonomischen Randbedingungen abzustimmen.

Anhand der Art der Kommunikationswege (Verbindungsleitungen zwischen den Rechnern) kann eine weitere Klassifizierung von Mehrrechnersystemen vorgenommen werden [9]:

- Systeme mit individuellen Kommunikationswegen
  Hierbei sind die einzelnen Rechner über fest zugeordnete Verbindungsleitungen miteinander gekoppelt. Zu dieser Klasse gehören Ring-, Maschennetz-, und Baumstrukturen. Beispiele hierfür sind in *Bild 2* dargestellt.

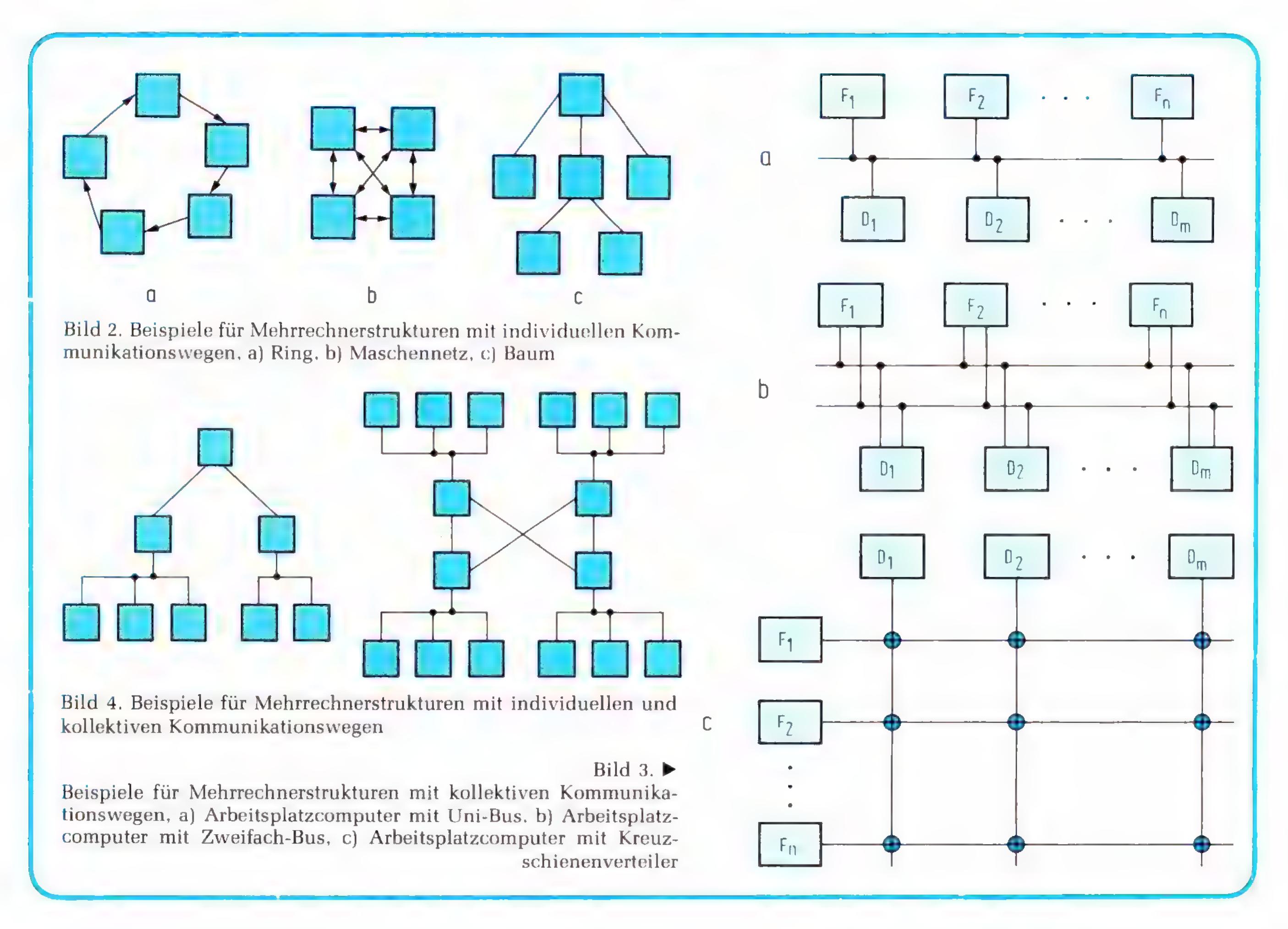


Bild 2. Beispiele für Mehrrechnerstrukturen mit individuellen Kommunikationswegen. a) Ring, b) Maschennetz, c) Baum

Bild 4. Beispiele für Mehrrechnerstrukturen mit individuellen und kollektiven Kommunikationswegen

Bild 3. ▶
Beispiele für Mehrrechnerstrukturen mit kollektiven Kommunikationswegen. a) Arbeitsplatzcomputer mit Uni-Bus, b) Arbeitsplatzcomputer mit Zweifach-Bus, c) Arbeitsplatzcomputer mit Kreuzschienenverteiler

– Systeme mit kollektiven Kommunikationswegen

Hierbei erfolgt die Koppelung der Rechner über gemeinsam benutzte Verbindungsleitungen. Zu dieser Klasse gehören die Bus-Systeme, auf die nachfolgend näher eingegangen wird. Beispiele sind in *Bild 3* angegeben.

– Systeme mit kollektiven und individuellen Kommunikationswegen

Bei komplexen Mehrrechnersystemen findet man auch oftmals Mischungen aus den ersten beiden Struktur-Klassen. Beispiele finden sich in *Bild 4*.

3.2 Bus-Systeme

Die meisten Mehrrechnerstrukturen verwenden Bus-Systeme. Entsprechend der unterschiedlichen Anforderungen ergeben sich Bus-Systeme unterschiedlicher Leistungsfähigkeit und Komplexität. Drei typische Vertreter dieser Klasse sind |10|:

– Uni-Bus-Systeme

Hierbei erfolgt die gesamte Kommunikation zwischen den Rechnern über eine gemeinsame Sammelschiene. Damit kann mit relativ geringem Aufwand eine hohe Modularität und Flexibilität erreicht werden. Allerdings kann bei System-Erweiterungen der Bus zum Engpaß werden. Außerdem kommt das Auftreten eines Defekts in der Bus-Steuerung einem totalen System-Ausfall gleich. *Bild 3a* zeigt den Aufbau des Arbeitsplatzcomputers mit Uni-Bus als Kommunikationssystem.

– Multi-Bus-Systeme (*Bild 3b*)

Hierbei stehen für die Rechner-Rechner-Kommunikation mehrere Sammelschienen zur Verfügung. Damit erhöht sich die Sicherheit und Verfügbarkeit des Systems, allerdings auf Kosten eines höheren Aufwands, da außer der Mehrfach-Auslegung des Busses eine Steuerlogik für die Auswahl und Überwachung erforderlich ist. In Bild 3b ist der Aufbau des Arbeitsplatzcomputers mit Zweifach-Bus als Kommunikationssystem dargestellt.

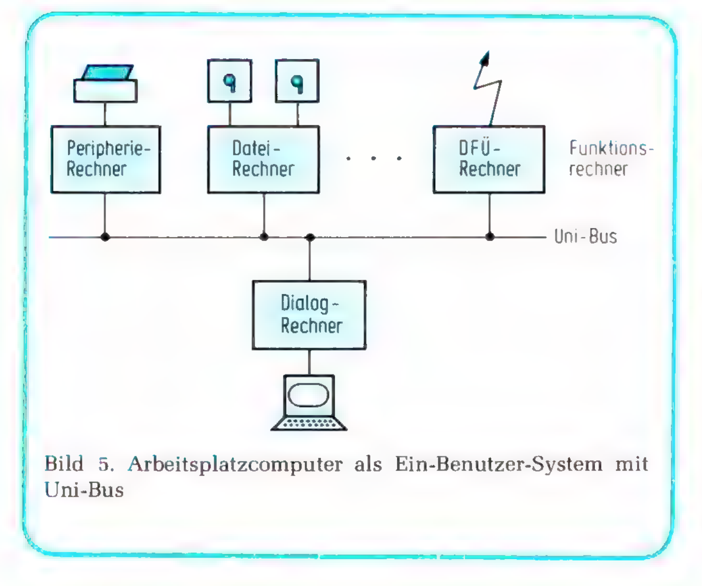


Bild 5. Arbeitsplatzcomputer als Ein-Benutzer-System mit Uni-Bus

– Systeme mit Kreuzschienenverteiler (Crossbar)

Hierbei sind m Systemelemente eines Typs (z. B. Dialogrechner) mit n Systemelementen eines anderen Typs (z. B. Funktionsrechner) matrixartig miteinander verbunden. Damit können bis zu min (m, n) Transfers gleichzeitig erfolgen. Jeder der m x n-Kreuzungspunkte beinhaltet eine Schalterfunktion und eine Auswahl-Logik, um bei Mehrfach-Anforderungen Bus-Konflikte zu verhindern. Dies macht einen erheblichen Aufwand erforderlich. *Bild 3c* zeigt den Aufbau des Arbeitsplatzcomputers als System mit Kreuzschienenverteiler.

4 Implementierung einer Uni-Bus-Struktur für den Arbeitsplatzcomputer

In den nachfolgenden Ausführungen wird ein Lösungsvorschlag zur Implementierung eines Uni-Bus-Systems vorgestellt. Als Beispiel dient der Arbeitsplatzcomputer in Form eines Einzelplatzsystems (single user system) mit dem in *Bild 5* dargestellten Aufbau. Um bei der Rechner-Rechner-Kommunikation möglichst hohe Übertragungsraten erzielen zu können, wird als Datenaustausch-Verfahren der hardwaregesteuerte direkte Speicherzugriff gewählt.

Der Kern des Kommunikationssystems ist der programmierbare DMA-Controller. Die Programmierung des DMA-Steuerbausteins, bei der diesem wichtige Informationen für den Datenaustausch (z. B. Blockanfangsadresse, Blocklänge, Übertragungsrichtung) mitgeteilt werden, obliegt nur dem Dialogrechner. Er kann auf Wunsch mit jedem der Funktionsrechner, die völlig gleichberechtigt sind, Daten austauschen. Damit die Adreßbereiche für den Datenaustausch in Sende- und Empfangsrechner voneinander unabhängig sind, ist für jeden Funktionsrechner eine Adreßgeberlogik vorgesehen, die vom Funktionsrechner programmiert werden kann.

In *Bild 6a* ist der Zustand des DMA-Busses für eine Datenübertragung vom Dialogrechner (DR) zu Funktionsrechner 1 (FR1) dargestellt. Dabei wird der Steuerbus von Sender und Empfänger vom DMA-Controller mit den für die Speicherzugriffe erforderlichen Signalen versorgt. Die Adreßbusse von Sender und Empfänger sind nicht miteinander verbunden, damit Daten aus einem beliebigen Quellbereich in einen beliebigen Zielbereich transferiert werden können. Für den Transfer müssen die beiden Datenbusse miteinander verbunden und in der entsprechenden Richtung durchgeschaltet sein.

Eingeleitet wird der Datenaustausch durch folgenden Mechanismus (*Bild 6b*): Für den Verbindungsaufbau erfolgt zunächst die Programmierung des DMA-Controllers und die Prüfung, ob FR 1 verfügbar ist. Falls ja, wird vom DR der Systembus von FR 1 angefordert ①. Sobald der DMA-Bus über den Systembus von FR 1 verfügen kann ②, fordert der DMA-Controller den Systembus von DR an ③. Nach Quittungsempfang ④ kann mit der Kommunikationsphase begonnen werden.

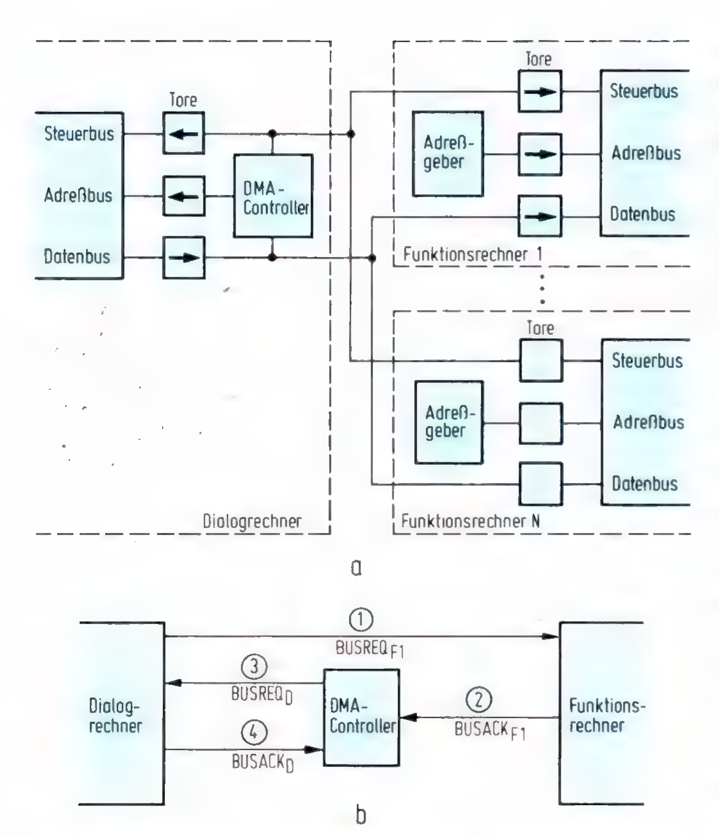


Bild 6. Datenaustausch beim Arbeitsplatzcomputer über DMA-Bus am Beispiel einer Übertragung vom Dialogrechner zum Funktionsrechner 1, a) Kommunikationsphase, b) Initialisierungsphase

Die Kommunikationsphase läuft in folgenden Schritten ab:

Zunächst wird vom DMA-Controller die Anfangsadresse des zu übertragenden Datenblocks auf den Adreßbus vom DR gelegt und ein Speicherlesesignal generiert. Damit gelangt das erste Datum auf den Datenbus von DR und FR 1. Außerdem wird von dem zu FR 1 gehörenden Adreßgeber die Zieladresse auf den Adreßbus von FR 1 gelegt. Nun wird vom DMA-Controller ein Speicherschreibsignal erzeugt und an FR 1 gesendet, womit die am Datenbus anliegende Information in der adressierten Speicherzelle angelegt wird. Danach wird Quell- und Zieladresse inkrementiert und das nächste Datum übertragen. Dieser Vorgang wiederholt sich solange, bis der gesamte Datenblock übertragen ist.

Der Datenaustausch wird abgeschlossen durch den Verbindungsabbau und die Generierung eines Unterbrechungssignals für FR 1, womit ihm der Empfang des Datenblocks gemeldet wird.

Ein Nachteil dieser Bus-Struktur liegt darin, daß die Programmierung des DMA-Controllers nur vom Dialogrechner aus vorgenommen werden kann und dadurch eine Master-Slave-Struktur entsteht. Zur Beseitigung dieses Nachteils kann der Bus-Steuerung ein eigener Rechner (Bus-Rechner) zugeordnet werden, dessen Aufgabe darin besteht, Kommunikationsanforderungen zu erkennen und zu behandeln. Die Existenz eines Bus-Rechners kann z. B. auch verwendet werden, um bei der Auftragsvergabe in Abhängigkeit vom Auftragstyp einen geeigneten Funktionsrechner zu ermitteln oder einen bestimmten Funktionsrechner auf dessen Bereitschaft zur Auftragsannahme zu prüfen oder bestimmte Maßnahmen zur Fehlererkennung und -behandlung vorzunehmen. Außerdem kann er für gewisse organisatorische Aufgaben herangezogen werden, beispielsweise zur Verhinderung von Verklemmungssituationen. Es ist jedoch wichtig, darauf zu achten, daß der Charakter eines völlig dezentral arbeitenden Mehrrechnersystems erhalten bleibt.

Literatur

1 *Flynn, M. J.*: Very high-speed computing systems. Proc. IEEE, vol. 54, (1966), H. 12, S. 1901...1909.
2 *Baer, J.-L.*: Multiprocessing systems. IEEE Transactions on computers, vol. C-25, (1976), H. 12, S. 1271...1277.
3 *Weissberger, A. J.*: Analysis of multiple-microprocessor system architectures. Computer Design, vol. 16, (1977), H. 6, S. 151...163.
4 *Raphael, H. H.*: Join micros into intelligent networks. Electronic Design, vol. 5, (1975), March 1, S. 52...57.
5 *Spetz, W. L.*: Microprocessor networks. Computer, vol. 10, (1977), H. 7, S. 64...70.
6 *Toong, H. D.*: Multi-microprocessor systems. Siemens Forschungs- u. Entwicklungsberichte, Bd. 7, (1978), H. 6, S. 311...315.
7 *Sauer, A.*: Sequential system structures. Siemens Forschungs- u. Entwicklungsberichte, Bd. 7, (1978), H. 6, S. 319...321.
8 *Bellm, H.; Sauer, A.*: Methods of data exchange between microcomputers, 3rd Euromicro Symposium, Amsterdam, Oct. 1977 (Preprints), S. 16...22.
9 *Engelhardt, H.; Feger, O.*: Multi-Microcomputersysteme. ELEKTRONIK, Bd. 27, (1978), H. 11, S. 49...53.
10 *Searle, B. G.; Freberg, D. E.*: Microprocessor applications in multiple processor systems. Computer, vol. 8, (1975), H. 10, S. 22...30.
11 *Neunert, H.*: Systemtechnik mit Mikrocomputern. ELEKTRONIK 1974, H. 10, S. 391...395.
12 *Kuhn, K.*: Mehrrechner- und echte Multiprozessor-Systeme mit fertigen µP-Platinen. ELEKTRONIK 1979, H. 8, S. 77...80.



Ing. (grad.) Hans Thinschmidt (links) ist in Käsmark geboren. Er studierte an der Fachhochschule Würzburg Allgemeine Elektrotechnik. 1960 trat er ins Zentrallabor der Siemens AG ein und beschäftigte sich auf dem Gebiet der Vermittlungstechnik, Datentechnik und Mikrocomputertechnik.
Hobbys: Tanzsport, Musik, Radfahren.
Telefon (0 89) 67 82-42 58

Dipl.-Inf. Hans Bellm (rechts), geboren in Weiher, studierte an der Technischen Universität Karlsruhe Informatik und trat 1975 in die Siemens AG ein. Dort arbeitet er seitdem auf dem Gebiet der Mikrocomputer-Strukturen.
Hobbys: Turnen, Reisen.
Telefon (0 89) 67 82-42 56

Dipl.-Ing. Frank Schmidtke, Dr.-Ing. Gerd Sandweg,
Dipl.-Inform. Hans Bellm

Ein Multi-Mikrocomputer-System am Arbeitsplatz

3. Teil: Verteiltes Betriebssystem eines Mehrrechnersystems

Für ein interaktives Mehrrechnersystem wird ein verteiltes Betriebssystem beschrieben, das an die jeweilige Einsatzanforderung angepaßt werden kann. Es ist sowohl ein Ein-Benutzer-, wie auch ein Mehr-Benutzer-System mit beliebig vielen Hintergrundrechnern möglich, wobei die anfallenden Aufträge parallel bearbeitet werden.

Die Hardware besteht aus einer beliebigen Anzahl von Rechnermodulen, die über ein Bussystem gekoppelt sind. Diese werden nach ihrem Aufgabenspektrum unterteilt in Dialog- und Funktionsrechner. Die Dialogrechner überwachen die Ausführung der Anwenderprogramme und steuern den Dialog mit den Anwendern über angeschlossene Bildschirmeinheiten. Außerdem verteilen sie Aufträge an die Funktionsrechner, die diese dann in eigener Regie ausführen.

Bei der Struktur und den Hauptmerkmalen des Betriebssystems steht die Forderung nach Modularität und Anpassung an flexible Hardware im Vordergrund. Dies wird vor allem erreicht durch die Bereitstellung von Betriebssystem-Bausteinen und eine Listenstruktur, die die aktuelle Systemkonfiguration repräsentiert.

1 Hardwarestruktur

Den Hardwareaufbau des Systems zeigt *Bild 1*. Sämtliche Rechner sind über ein Bussystem verbunden. Dabei verfügt jede Einheit über eine Verarbeitungseinheit (CPU) und einen eigenen Speicher (RAM) zur Aufnahme von Daten und Programmen, um die selbständige Bearbeitung von Funktionen zu gewährleisten. Die Zahl der anschließbaren Rechner ist variabel.

Die *Dialogrechner* (DR) stellen die Schnittstelle zwischen Benutzer und dem System dar. Sie haben die Aufgabe, Kommandos vom Benutzer zu interpretieren und auszuführen, Anwenderprogramme zu laden und deren Ausführung zu überwachen. Dabei ist jeder DR nur für das Programm eines Benutzers zuständig. Das heißt, diese Rechner arbeiten benutzerorientiert. Die hierbei anfallende Überwachungsfunktion beinhaltet vor allem die Verteilung von Systemfunktionen an die Funktionsrechner und deren Koordinierung.

Im Gegensatz zu den Dialogrechnern arbeiten die *Funktionsrechner* (FR) funktionsorientiert, d. h. sie erledigen für beliebige Auftraggeber innerhalb des Systems bestimmte Arbeiten. Diese Aufgaben können sowohl das Vorhandensein bestimmter, an diese Rechner angeschlossener Peripheriegeräte voraussetzen, als auch reine Verarbeitungsfunktionen beinhalten. Die Bearbeitung dieser Aufträge erfolgt selbständig, solange die benötigten Betriebsmittel direkt verfügbar sind. Anderenfalls können Aufträge an andere Funktionsrechner delegiert werden.

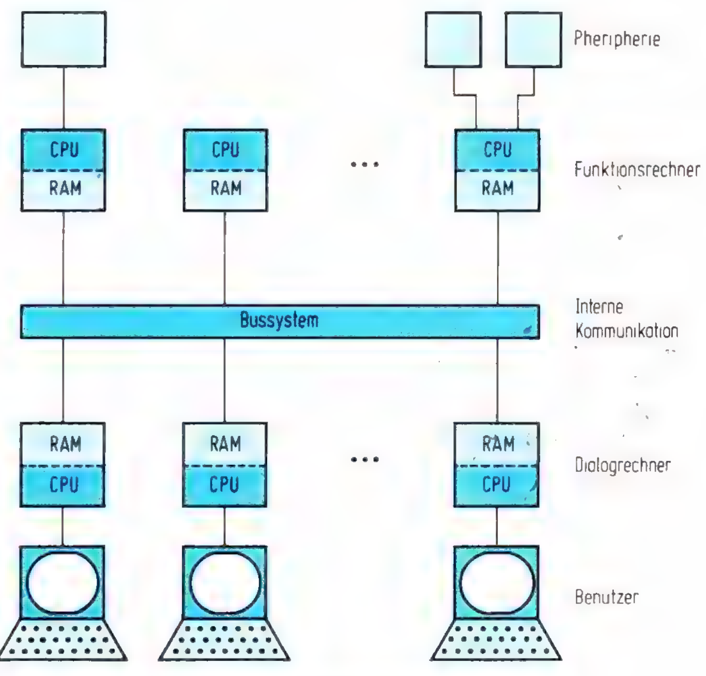


Bild 1. Struktur des Mehrrechnersystems

Über ein internes *Bussystem* kommunizieren die Rechnereinheiten untereinander. Dort werden Aufträge, Daten und Zustandsmeldungen übertragen. Der Datenaustausch wird über Direct Memory Access (DMA) vorgenommen (siehe Teil 2). Dabei handelt es sich um eine sehr schnelle parallele, rein hardwaregesteuerte Datenübertragung.

Der Datenaustausch zwischen den Rechnern wird über Meldungen abgewickelt. Der Empfänger wird zur Übernahme der Meldung in seinen laufenden Arbeiten unterbrochen. Das ermöglicht trotz selbständiger Arbeitsweise der Einzelrechner beliebige Kommunikationen zu beliebigen Zeitpunkten.

2 Merkmale des Betriebssystems

Das Betriebssystem (BS) ist für unterschiedliche Konfigurationen und Leistungsanforderungen konzipiert. Es soll sowohl Ein- als auch Mehrbenutzerbetrieb ermöglichen und eine beliebige Zahl von Funktionsrechnern und Peripheriegeräten zulassen. Hierzu wird aus BS-Modulen ein individuelles Betriebssystem erstellt, das den jeweiligen Gegebenheiten angepaßt ist.

Die Flexibilität des Systems wird erreicht durch strenge Strukturierung und die Definition klarer Schnittstellen zwischen den einzelnen Betriebssystem-Bausteinen. Die Verteilung der Aufträge auf die einzelnen Rechner erfolgt ausschließlich über Systemlisten. Diese werden bei der Generierung des Systems erstellt und repräsentieren die aktuelle Konfiguration. Sie enthalten Angaben über angeschlossene Rechner, deren Peripheriegeräte und das Spektrum der von jedem Rechner ausführbaren Funktionen.

Parallelarbeit im System wird realisiert zum einen durch Einführung eines Mehrbenutzerbetriebs mit mehreren Dialogrechnern und zum anderen durch die Parallelisierung von Teilaufgaben der Anwenderprogramme.

Die Parallelisierung innerhalb einzelner Anwenderprogramme erfolgt durch die Delegierung von Aufträgen an die Funktionsrechner in Form von Systemaufrufen. Dabei wird zwischen Basis-Systemdiensten und Anwender-Systemdiensten unterschieden.

Unter einem *Basis-Systemdienst* (BSD) wird eine Betriebssystemfunktion verstanden, die zur Bedienung von Peripheriegeräten, Dateiverwaltung oder Ablaufsteuerung dient. Wesentlich ist, daß diese Systemdienste von einem Rechner ohne die Hilfe anderer erledigt werden können. Die dazu benötigten Systemroutinen sind resident gespeichert. BSDs sind z. B. Eingabe- und Ausgabeaufträge, Öffnen und Schließen von Dateien sowie Warten auf Rückmeldungen von Aufträgen. Beim Erkennen solcher Systemaufrufe wird analysiert, ob der entsprechende Rechner die gewünschte Funktion selbst ausführen kann, oder ob dieser Auftrag an einen anderen Rechner delegiert werden muß.

Anwender-Systemdienste (ASD) können vom Anwender, unterstützt vom Betriebssystem, selbst definiert werden. Sie haben den Zweck, bestimmte Funktionen eines Anwenderprogramms auf beliebige Funktionsrechner auszulagern, die diese parallel zu anderen Programmteilen ausführen. Im Unterschied zum BSD kann ein ASD von jedem beliebigen Rechner bearbeitet werden. Ein weiteres Merkmal der ASDs besteht darin, daß sie nicht zum residenten Teil des Betriebssystems gehören. Die ASD-Routinen werden vielmehr bei Bedarf von einem externen Speicher geladen, um Speicherplatz in den Funktionsrechnern einzusparen.

Beispiele solcher ASDs sind das Durchsuchen von Dateien, Formatierung von Daten oder die Abwicklung eines Datenaustausches mit einem Großrechner.

3 Arbeitsweise des Systems

Jeder Dialogrechner übernimmt die Ausführung eines Anwenderprogramms einschließlich Benutzerdialog und delegiert gegebenenfalls die im Programm aufgerufenen Systemdienste an die Funktionsrechner. Basis-Systemdienste, die Peripheriegeräte ansprechen, werden ausschließlich an Funktionsrechner delegiert, die diese Geräte steuern, während Anwender-Systemdienste nach vorgegebenen Strategien (z. B. erster freier FR, der am wenigsten ausgelastete FR) vergeben werden.

Um nach der Auftragsvergabe die Programmbearbeitung fortsetzen zu können, wird ein delegierter Auftrag in einer Tabelle festgehalten. Dadurch werden zeitliche Überwachung, spätere Synchronisierung und richtige Zuordnung von eventuell eintreffenden Rückmeldungen des betreffenden Auftrags ermöglicht. Die Aufnahme in die Tabelle ist allerdings nur dann notwendig, wenn von einem Auftrag Ergebnisse zurückerwartet werden und bis zum Eintreffen dieser Ergebnisse andere Arbeiten zu erledigen sind. Andernfalls gibt der Auftraggeber die Verantwortung an den ausführenden Funktionsrechner ab und setzt seine Arbeiten ohne weitere Bezugnahme auf den delegierten Auftrag fort. Eine Rückmeldung erfolgt lediglich im Fehlerfall.

Ein FR kann zwei Typen von Aufträgen bearbeiten: Anwender-Systemdienste und Basis-Systemdienste. Art und Umfang der ausführbaren BSDs werden durch die angeschlossene Peripherie bestimmt, während hinsichtlich der ASDs keine Einschränkungen bestehen. Da alle Rechner selbständig arbeiten und Aufträge zu beliebigen Zeitpunkten eintreffen können, ist es erforderlich, die eingehenden Aufträge in einer Warteschlange zwischenzuspeichern, um beim Auftraggeber keine unnötigen Wartezeiten entstehen zu lassen.

Die in die Warteschlange aufgenommenen Aufträge werden entweder nach der Reihenfolge ihres Eintreffens (first in – first out), oder nach Dringlichkeit (Priorität) abgearbeitet.

In *Bild 2* wird die Arbeitsweise des Systems anhand eines schematisierten Ablaufs eines Benutzerprogramms veranschaulicht. Als Hardwarekonfiguration wird z. B. ein System mit einem Dialogrechner und zwei Funktionsrechnern vorausgesetzt. Dabei werden nur die für das Zusammenspiel der einzelnen Rechner wichtigen Programmteile markiert. Zur Identifizie-

rung werden die einzelnen Aufträge fortlaufend durchnumeriert.

4 Struktur des Betriebssystems

Das gesamte Betriebssystem ist so strukturiert, daß jedem Rechner jeweils nur die Funktionen zugeordnet werden, die seinem Aufgabenspektrum entsprechen (*Bild 3*).

Das Basis-BS enthält Routinen, die auf allen Rechnern verfügbar sein müssen. Dazu gehören Auftragsverwaltung, Kommunikationssteuerung sowie Analyse und Verteilung von Systemdiensten.

Spezifisch für die Dialogrechner sind die Routinen zur Interpretation und Ausführung von Benutzerkommandos, wozu auch das Laden und die Ausführungsüberwachung von Anwender- und Dienstprogrammen gehört. Kommandos und Eingaben können auch im Stapelbetrieb verarbeitet werden. Außerdem müssen diese Rechner die Behandlung von System-Fehlermeldungen vorsehen.

Die Funktionsrechner enthalten Routinen zur Auftragsannahme, Einrichtung und Bearbeitung von Warteschlangen, Speicherverwaltung, Ausführung von Basis-Systemdiensten, Laden und Ausführung von Anwender-Systemdiensten sowie die Steuerung der jeweils angeschlossenen Peripheriegeräte.

Die oben beschriebenen Betriebssystem-Teile werden bei der Systeminitialisierung geladen und bilden das statische (residente) Betriebssystem der jeweiligen Rechner. Die Anwender-Systemdienste dagegen können bei Bedarf von jedem Rechner geladen werden

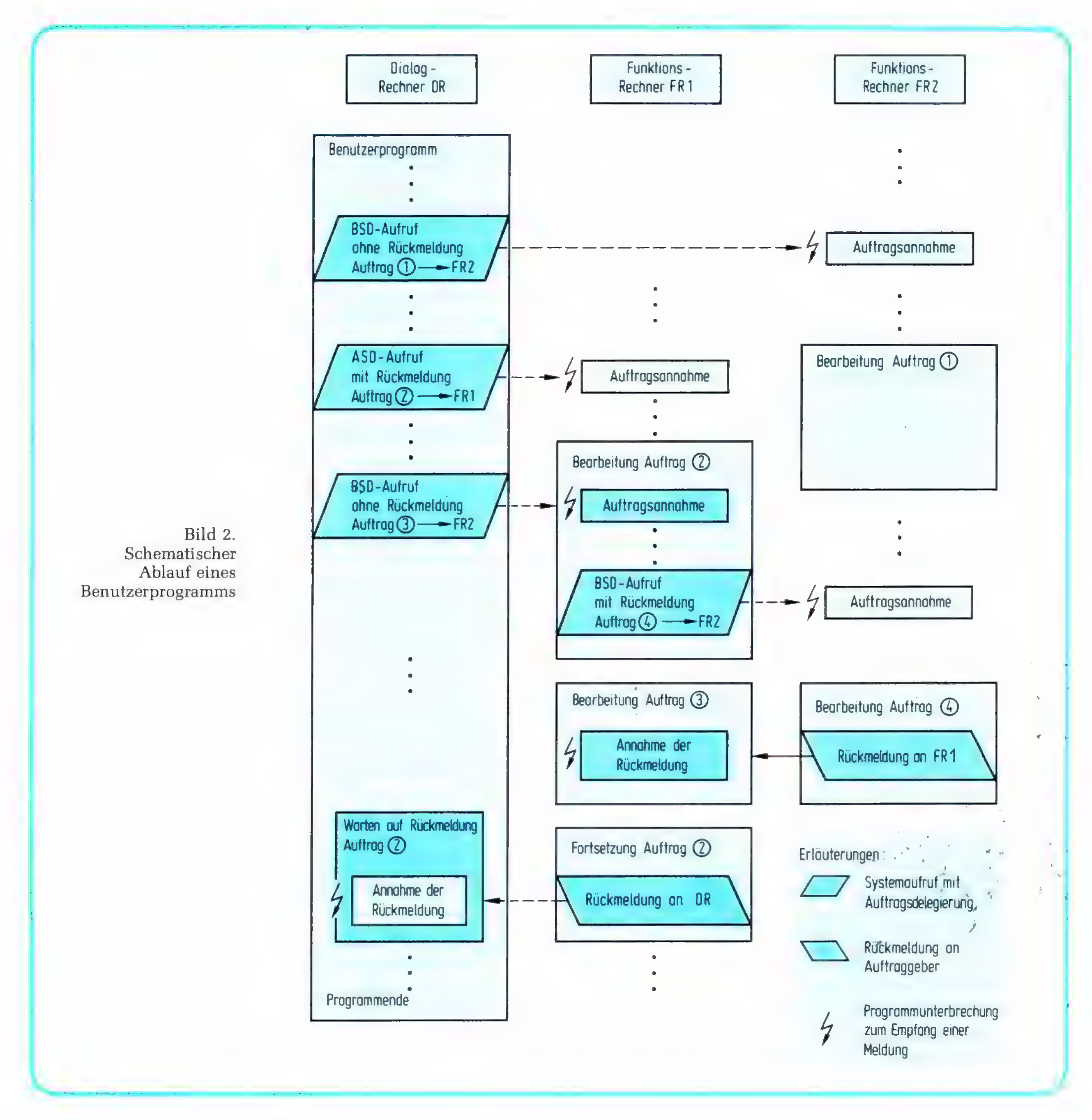


Bild 2. Schematischer Ablauf eines Benutzerprogramms

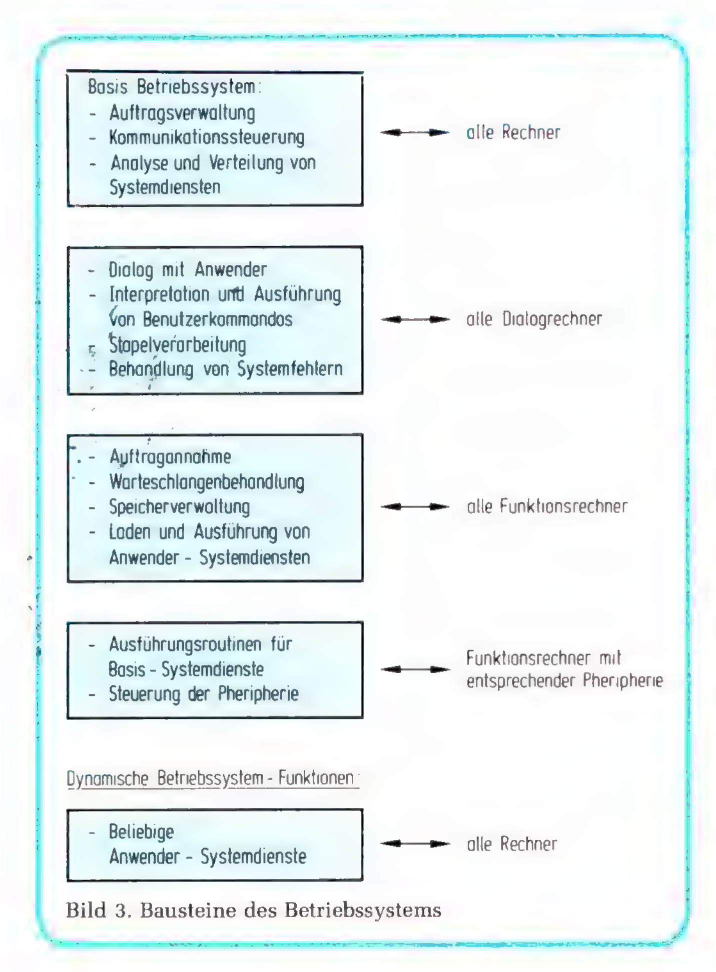


Bild 3. Bausteine des Betriebssystems

und gehören somit zum dynamischen (nicht residenten) Teil des Betriebssystems.

5 Systemgenerierung

Unter Systemgenerierung wird das Zusammenstellen der aktuellen Betriebssystem-Software für eine spezielle Rechnerkonfiguration verstanden. Bei der Systeminbetriebnahme wird die Zahl der Dialogrechner und Funktionsrechner sowie die Zuordnung von Peripheriegeräten zu diesen Rechnern angegeben. Aus diesen Angaben werden die entsprechenden Betriebssystem-Moduln (s. Bild 3) für die einzelnen Komponenten zusammengestellt und die Systemlisten aufgebaut, die diese Konfiguration wiedergeben. Bei der Systeminitialisierung lädt jeder Rechner diese Listen und die für ihn relevanten Teile des Betriebssystems.

Aufgrund der listenorientierten Arbeitsweise des Systems kann auch im Betrieb eine Umkonfigurierung vorgenommen werden. Dies ist z. B. nach dem Erkennen von fehlerhaften Komponenten sinnvoll. Durch Änderung der Listen werden fehlerhafte Rechner stillgelegt und deren Funktionen anderen Einheiten übertragen. Die nicht beendeten Aufträge werden von den jeweiligen Auftraggebern neu vergeben.

6 Betriebssystem-Schnittstelle zum Anwender

Das Betriebssystem bietet dem Anwender Hilfen, die ihm ein einfaches Arbeiten mit einem solchen Mehrrechnersystem ermöglichen. Eine wichtige Forderung dafür ist, daß der Benutzer keine Rücksicht auf die aktuelle Hardwarekonfiguration nehmen muß. Das gilt sowohl für die Kommandosprache als auch für die Programmierung. Weiterhin müssen Software-Testhilfen, wie Monitor, Tracer u. ä. zur Verfügung stehen, die insbesondere auch das Zusammenspiel der einzelnen Rechner berücksichtigen.

Die Programmierung paralleler Prozesse wird erleichtert durch einfache Unterprogrammaufrufe (Systemdienstaufrufe). Für einige Parameter können gegebenenfalls die vom System vorgegebenen Standards eingesetzt werden. Dies ermöglicht dem Programmierer ein stufenweises Einarbeiten in die Möglichkeiten des Systems.

Schließlich kann der Benutzer Teilroutinen oder Unterprogramme definieren, die das Betriebssystem als Anwender-Systemdienste übernimmt. Damit kann er sich selbst die Hilfsmittel zur Parallelisierung seines Problems schaffen.

Literatur

[1] *Goos, G.; Hartmanis, J.*: Parallel Processing. Lecture Notes in Computer Science. Springer Verlag 1975.

[2] *Wettstein, H.*: Aufbau und Struktur von Betriebssystemen. Hanser Verlag, 1978.

[3] *Zima, H.*: Betriebssysteme – Parallele Prozesse. BI-Wissenschaftsverlag, 1976.

[4] *Kartashev, S. I.; Kartashev, S. P.*: Dynamic Architectures: Problems and Solutions. Computer, July 1978, S. 26...40.

[5] INTEL Corporation: RMX/80 USER's GUIDE Manual Nr. 9800522B, 1977.

[6] *Sauer, A.*: Sequential System Structures. Siemens Forschungs- und Entwicklungsberichte, Bd. 7 (1978), H. 6, S. 319...321.

[7] *Kroneberg, A.*: Hierarchisch strukturierte System-Software. Siemens Forschungs- und Entwicklungsberichte, Bd. 7 (1978), H. 6, S. 348...351.

[8] *Thompsen, K.*: UNIX Implementation. Bell System Technical Journal, July-August 1978, S. 1931...1946.

[9] *Ritchie, D. M.*: UNIX Time-Sharing System – A Retrospective. Bell System Technical Journal July-August 1978, S. 1947...1969.

[10] *Gonzales, M. J.; Ramamoorthy, C. V.*: Parallel Task Execution in a Decentralized System. IEEE Transaction on Computers, VOL. C-21, H. 12, December 1972, S. 1310...1322.

[11] *Bellm, H.; Sauer A.*: Methods of data exchange between microcomputers. 3rd Euromicro Symposium, Amsterdam, October 1977 (Preprints), S. 16...22.

[12] *Will, B.*: Erhöhte Ausfallsicherheit bei Systemen verteilter Intelligenz. ELEKTRONIK 1979, H. 7, S. 69...73.



Dipl.-Ing. Frank Schmidtke (links) ist gebürtiger Rostocker. Er studierte Nachrichtenverarbeitung an der TH Karlsruhe. 1978 trat er in die Siemens AG ein und beschäftigt sich mit Betriebssystemen für Multi-Mikrorechnerstrukturen.
Hobbys: Reisen, Musik
Telefon (0 89) 67 82-42 57

Dr.-Ing. Gerd Sandweg (rechts) ist in Stuttgart geboren. Er studierte Nachrichtentechnik an der TH München. 1978 trat er in die Siemens AG ein und beschäftigt sich mit Rechnerarchitekturen.
Hobbys: Sport, Basteln
Telefon (0 89) 67 82-33 13

Dipl.-Inform. Hans Bellm wurde im 2. Teil vorgestellt.

Dr.-Ing. Gerd Sandweg, Dipl.-Ing. Frank Schmidtke

Ein Multi-Mikrocomputer-System am Arbeitsplatz

4. Teil: Praktische Einsatzbeispiele eines Multi-Mikrocomputer-Systems

Ein Mehrrechnersystem kann aufgrund seiner Flexibilität einen großen Aufgabenbereich abdecken, der von Minicomputeraufgaben bis zu Großrechnersystemleistungen reicht. Dieses Einsatzspektrum soll hier skizziert werden. Außerdem sind drei Programmbeispiele beschrieben, die die Dialogführung, Datenfernverarbeitung und ein Datenbanksystem betreffen.

In den vorausgegangenen Aufsätzen dieser Reihe wurden Konzept und Zielsetzung eines Multi-Mikrocomputer-Systems vorgestellt. Nun werden Einsatzmöglichkeiten eines solchen Systems skizziert und drei Anwenderprogrammpakete beschrieben. Sie betreffen die Dialogführung, Datenfernverarbeitung und ein Datenbanksystem. Die Programme wurden in erster Linie als Test- und Demonstrationsbeispiele entwickelt und sollten nicht mit kommerziellen Programmsystemen verglichen werden. Die vorgestellten Programmpakete wurden auf einem Mehrrechnersystem mit vier Mikrocomputern erstellt und ausgetestet.

1 Einsatzspektrum

Ein Ziel eines Multi-Mikrocomputer-Systems liegt darin, durch Parallelarbeit von standardisierten, billigen Hardwarekomponenten ein besseres Preis/Leistungsverhältnis als bei einem hochgezüchteten Einzelrechner zu ermöglichen. Dieses Ziel ist nur dann erreichbar, wenn die zu bearbeitenden Aufgaben groß genug und parallelisierbar sind, um mehrere Rechner auslasten zu können. Bei einem Mehrrechnersystem ist eine Parallelarbeit auf Task- und auf Jobebene sinnvoll.

Unter Parallelarbeit auf Taskebene wird hier die gleichzeitige Abarbeitung von abgeschlossenen Teilaufgaben eines Programms auf mehreren Rechnern verstanden. Solche Teilaufgaben können z. B. die Bedienung von Peripheriegeräten oder Dateioperationen sein. Der Entwickler von Anwendersoftware sollte eine gegebene Aufgabe in möglichst parallel bearbeitbare Teile zerlegen. Dies ist schwierig, wenn sequentielle Abhängigkeiten bestehen (z. B. Compiler, Editor). Einfach ist dagegen die Aufteilung, wenn ein Programm aus funktionellen Blöcken besteht, die nur schwach verkoppelt sind, oder wenn große Datenmengen zu verarbeiten sind, die sich in Teilfelder gliedern lassen (z. B. Suchen, Sortieren, Bildverarbeitung, bestimmte Matrixoperationen). Eine weitere Maßnahme zur Steigerung der Effektivität eines Mehrrechnersystems ist die Realisierung der Parallelarbeit auf Jobebene. Dabei werden gleichzeitig Anwenderprogramme mehrerer Benutzer bearbeitet.

Unter Berücksichtigung der obrigen Überlegungen lassen sich folgende Rechnerkonfigurationen, Anwendungsbeispiele und Einsatzgebiete angeben:

Minimalkonfiguration mit zwei Rechnern:
Kleine Anwendungen, für die heute Minicomputer eingesetzt werden; Aufteilung in Hauptrechner und Ein-/Ausgaberechner bietet sich an.

Einbenutzersystem mit bis zu vier Rechnern:
Informationssystem, intelligentes Terminal, Auftragsverfolgung, Lagerverwaltung, Buchhaltung, Textverarbeitung; Einsatz in Sekretariat, Arztpraxis, Ingenieurbüro usw.

Mehrbenutzersystem mit vielen Rechnern:
Datenbanken, Verwaltung, technisch-wissenschaftliche Berechnungen; Einsatz in Büro, Klinik, Universität usw.

2 Dialoghilfen für Anwenderprogramme

Ein wichtiger Gesichtspunkt des verwendeten Mehrrechnersystems ist die dialogorientierte Arbeitsweise. Die Kommunikation des Benutzers mit einem Programm kann auf vier Arten erfolgen:

a) Strenger Dialog mit Frage und Antwort.

Der Programmieraufwand hierfür ist gering. Es muß immer nur eine Eingabe geprüft und im Fehlerfall lediglich die zugehörige Frage wiederholt werden.

Nachteilig an dieser Dialogart ist der geringe Überblick, der dem Benutzer geboten wird. Bei großen Programmen ist es für den Anfänger schwierig, durch viele Einzelentscheidungen an die richtige Stelle zu gelangen. Andererseits langweilen den geübten Benutzer die ausführlichen Fragestellungen und die vielen notwendigen Eingaben.

b) *Ausfüllen von Masken*

Diese Dialogart setzt ein Bildschirmgerät voraus. Auf ihm werden Eingabeformulare, sog. Masken, angeboten, deren Felder der Benutzer ausfüllen kann. Die Eingaben erfolgen damit parallel in einem größeren Zusammenhang und der Benutzer erhält einen besseren Überblick über Umgebung, Möglichkeiten und Auswirkungen einzelner Eingaben.

Nachteilig bei dieser Dialogart ist der zunächst relativ hohe Programmieraufwand. Der Aufbau von Masken und die Überprüfung mehrerer gleichzeitiger Eingaben, die sich eventuell widersprechen können, ist aufwendig. Um diese Nachteile zu beseitigen, stehen eine Reihe von Programmbausteinen zur Verfügung, die Definition, Ausgabe und Interpretation von Masken und Maskeneinträgen unterstützen. Ein größerer Baustein dieser Art ist ein sog. Maskengenerator. Mit diesem Programm und den hardwaremäßigen Editionshilfen eines Bildschirmgeräts kann direkt auf dem Bildschirm eine gewünschte Maske erstellt werden. Wenn das Maskenbild festgelegt ist, können später den einzelnen Eintragsfeldern noch Attribute zugeordnet werden, die Grenzwerte, Plausibilitätsbedingungen, Formate, Schutzfunktionen u. ä. betreffen.

c) *Kommandosprache*

Bei den beiden bisher beschriebenen Dialogformen liegt die Initiative beim Programm. Dies vereinfacht den Dialog auf Kosten der Flexibilität. Will jedoch der Benutzer die Initiative übernehmen und soll auch die Behandlung von Sonderfällen möglich sein, ist eine Kommandosprache das geeignete Hilfsmittel. Mit einer geringen Anzahl beliebig kombinierbarer Sprachelemente kann dann der geübte Benutzer alle Möglichkeiten eines Programms ausschöpfen.

Für die Syntaxprüfung und Interpretation solcher Kommandos gibt es Programmbausteine, mit deren Hilfe der Programmierer schnell eine problemangepaßte Kommandosprache definieren kann.

d) *Funktionstastatur*

Mit der Funktionstastatur kann der Benutzer zu beliebigen Zeitpunkten in den Programmablauf eingreifen. Dabei kann die Bedeutung der Funktionstasten sowohl vom System als auch vom Benutzerprogramm vorgegeben werden.

In der Praxis werden die vier genannten Dialogarten vermischt angewandt. Bei beiden im folgenden beschriebenen Programmsystemen sind z. B. sowohl Maskeneinträge als auch Kommandos möglich.

3 Datenfernverarbeitung

Eine wichtige Funktion eines Rechners am Arbeitsplatz ist der Anschluß an ein Großrechenzentrum. Der Benutzer erhält damit Zugang zu zentralen Daten, die er vor Ort bearbeiten kann. Dies trägt zu einer Entlastung und effektiveren Ausnutzung von Großrechner und Übertragungsleitung bei. Für das beschriebene Multi-Mikrocomputer-System stehen Programme zur Verfügung, die folgende Funktionen der Datenfernverarbeitung ermöglichen:

- Automatische Anwahl an ein Rechenzentrum,
- Anschluß an Großrechenanlage,
- Senden von Dateien,
- Empfangen von Dateien,
- Dialog mit Großrechenanlage,
- Abbau der Verbindung.

Zur Einleitung einer Datenfernverarbeitung muß der Benutzer zunächst eine Verbindung zu einer Großrechenanlage aufbauen. Er kreuzt dazu in einer Maske die gewünschte Rechenanlage an, die daraufhin automatisch angewählt wird. Falls keine Verbindung zustande kommt, können weitere Wählversuche unternommen werden. Ist eine Verbindung zum Großrechner hergestellt, kann in einer Maske das gewünschte LOGON-Kommando angekreuzt werden, welches an den Großrechner gesendet wird und den Anschluß an das Betriebssystem herstellt. Die Masken für die Auswahl der Telefonnummern und der LOGON-Kommandos definiert der Benutzer selbst. Änderungen können also ohne Programmieraufwand schnell berücksichtigt werden. Für Spezialfälle, die durch die gerade verfügbare Maske nicht abgedeckt werden, ist ein Eingabefeld vorgesehen, in das der Benutzer z. B. eine Telefonnummer oder ein besonderes LOGON-Kommando eingeben kann.

Wenn das Betriebssystem der Großrechenanlage sich meldet, kann der Benutzer Kommandos an die Großrechenanlage schicken und Antworten empfangen. Das Mehrrechnersystem arbeitet also online als Terminal an einer Großrechenanlage. Der Benutzer kann aber auch zentral gespeicherte Dateien abrufen, sie vor Ort offline bearbeiten und die Ergebnisse wieder zurückschicken. Alle Operationen und Eingaben werden soweit möglich bereits vor Ort geprüft. Die Programme für die Datenfernverarbeitung dienen also dem Komfort und der Sicherheit. Außerdem ist Parallelarbeit während den Datenübertragungsphasen möglich.

4 Datenbanksystem

Datenbanksysteme sind für Mehrrechnersysteme ein geeignetes Anwendungsbeispiel, da sich typische Funktionen wie Suchen und Sortieren relativ leicht parallelisieren lassen. Für das hier vorgestellte Mehrrechnersystem wurde ein kleineres Datenbanksystem entwickelt, bei dem besonderes Gewicht auf kurze Einarbeitungszeit und Benutzerfreundlichkeit gelegt wurde. Entsprechend einfach ist die Programmstruktur. Es gibt nur zwei Ebenen: Auswahlebene und Funktionsebene (*Bild 1*). Die einzelnen Funktionen sind durch Programmmodule realisiert, die völlig un-

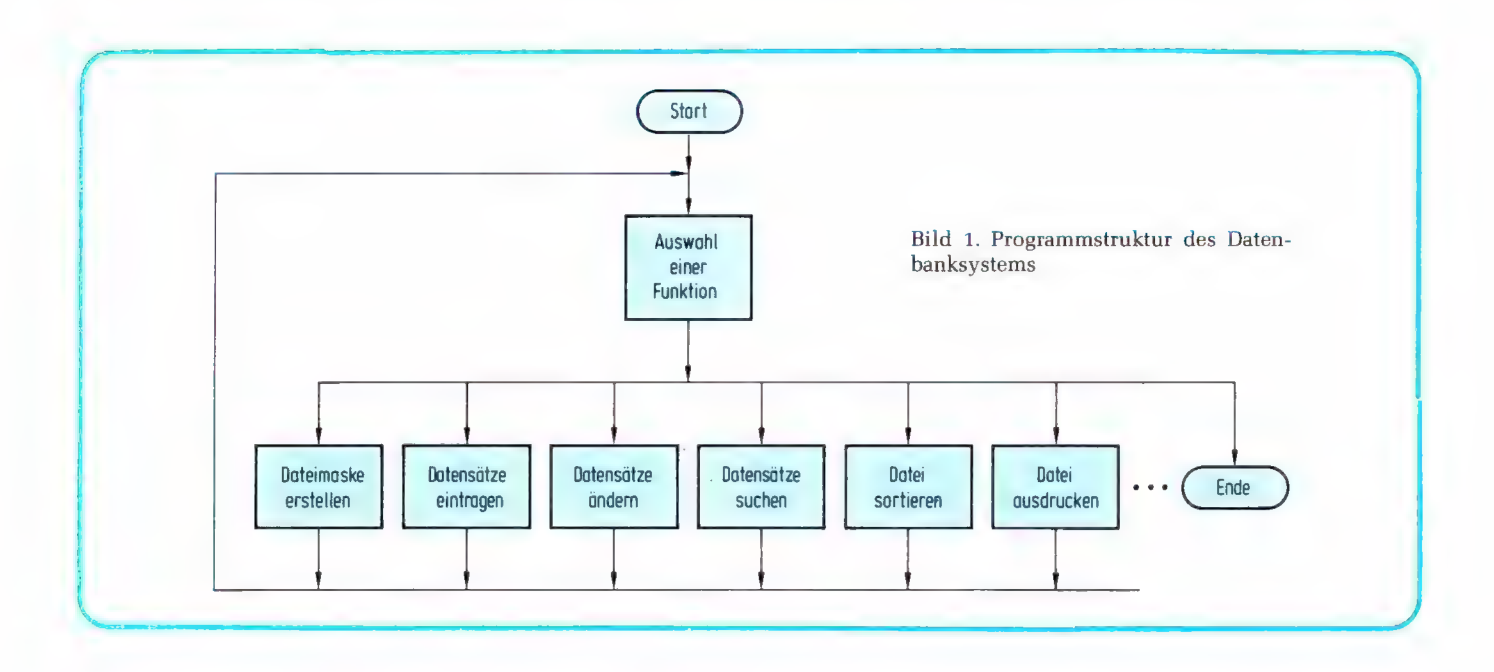


Bild 1. Programmstruktur des Datenbanksystems

abhängig voneinander sind. Jeder bearbeitet eine Datei und erzeugt gegebenenfalls eine Ergebnisdatei. Diese Ergebnisdatei kann von dem gleichen oder einem anderen Modul weiterverarbeitet werden usw.

Der Dialog des Benutzers mit den Modulen ist einheitlich und maskenorientiert aufgebaut. Der Anwender erstellt für jede Datei eine „Dateimaske", die einerseits die Struktur der Datensätze definiert und andererseits für Ein- und Ausgaben von Datenbankfunktionen dient.

Will der Benutzer z. B. eine Datei nach bestimmten Begriffen durchsuchen, füllt er die Eintragsfelder der Dateimaske mit den Zeichenfolgen aus, nach denen in den Datensätzen gesucht werden soll (*Bild 2*). Hierbei kann angegeben werden, ob die Zeichenfolge im betreffenden Satzteil enthalten (Normalfall), identisch (=), größer (>, bezogen auf alphabetische Ordnung) oder kleiner (<) sein soll. Wenn mehrere Felder der Dateimaske ausgefüllt werden, wird dies als eine UND-Verknüpfung der Bedingungen verstanden.

Diese Regeln sind sofort einsichtig und erlauben die Bearbeitung der häufigsten Fälle. Für Sonderfälle, z. B. eine ODER-Verknüpfung, kann unter die Maske eine Kommandozeile geschrieben werden. Diese Kommandozeile kann in beliebiger Zahl die Feldnamen (bzw. Abkürzungen), die Vergleichsfunktionen, die Vergleichswerte und die Verknüpfungsoperatoren enthalten. Die Eingaben bei den anderen Funktionen erfolgen in ähnlicher Weise.

Für Programmverzweigungen und spezielle funktionsabhängige Eingaben wird bei jedem Programmodul eine sog. „Steuermaske" der Dateimaske vorangestellt. Diese Steuermaske enthält ein Eintragsfeld zur Angabe, ob evtl. eine Bedienungsanleitung oder eine Beendigung des Moduls gewünscht wird. Außerdem enthält die Steuermaske meist zwei Eintragsfelder für die Namen der zu bearbeitenden Datei und der Ergebnisdatei.

Das bestehende Datenbanksystem eignet sich aufgrund seiner Modularität gut für die Realisierung von Parallelarbeit in dem beschriebenen Mehrrechnersystem. So kann bei der Funktion „Suchen" eine Datei in Teildateien zerlegt werden, die von mehreren Rechnern gleichzeitig durchsucht werden. Bei der Funktion „Sortieren" können mehrere Rechner Teildateien sortieren und danach ein Rechner die sortierte Gesamtdatei erstellen. Eine andere Art der Parallelarbeit ergibt sich bei Funktionen, deren Ende nicht abgewartet werden muß, wie Ausgaben auf Peripheriegeräte. Nach der Delegierung solcher Funktionen können sofort weitere Aufträge bearbeitet werden. Durch Einführung eines Mehrbenutzerbetriebs läßt sich die Auslastung des Mehrrechnersystems noch verbessern. Hierbei sind allerdings Fragen der Zugriffsberechtigung und des Datenschutzes zu beachten. Diese Aspekte sind beim Konzept des Datenbanksystems von vorneherein berücksichtigt, damit entsprechende Erweiterungen schrittweise vorgenommen werden können.

Literatur

Siehe vorhergehende Teile.

Die Autoren wurden im 3. Teil vorgestellt.

SUCHEN (Y) / HILFE (H) / ENDE (E) : Y
QUELLDATEI : PERSON ZIELDATEI: PERSON.1
NAME : VORNAME:
STRASSE :
ORT : MUENCHEN
TELEFON :
BERUF : ANGESTELLTER

Bild 2. Beispiel einer ausgefüllten Maske. Der hier formulierte Datenbankauftrag lautet: „Suche aus der Datei PERSONEN alle Sätze, die im Feld ORT das Wort MUENCHEN und im Feld BERUF das Wort ANGESTELLTER enthalten. Die gefundenen Sätze sollen in der Datei PERSON. 1 abgelegt werden."

Dipl.-Ing. Anton Sauer, Dipl.-Ing. Eduard Scheiterer,
Ing. (grad.) Hans Thinschmidt

Ein Multi-Mikrocomputer-System am Arbeitsplatz

5. Teil: Entwicklung, Test und Aufbau von Mehrrechnersystemen auf Mikrocomputerbasis

In den vorausgegangenen Teilen dieser Serie wurden generelle und detaillierte Aussagen über Prinzip, Sinn, Ablauf und einige Anwendungen des Arbeitsplatzcomputers auf der Basis einer Multi-Mikrocomputerstruktur gemacht. Dieser Beitrag berichtet über eingesetzte Test- und Prüfverfahren dieses Systems. Darüber hinaus werden Hinweise für die praktische Realisierung und Konstruktion derartiger Rechner gegeben, die erprobt wurden und damit zur Nachahmung empfohlen werden.

1 Testmöglichkeiten

Testmöglichkeiten von hochintegrierten Bausteinen bei der Herstellung und im Betrieb sind entweder unzureichend oder zu aufwendig. Es sind verschiedene Möglichkeiten denkbar, dieses zu beheben [1]:

- Einbeziehung von Zufallstestmustern und algorithmisch erzeugten Tests in Funktionstests.
- Zusammenarbeit zwischen Hersteller und Anwender, um einen besser testbaren Schaltungsentwurf zu erzielen.

Auf der Chipebene scheint dieser zweite Punkt in absehbarer Zeit nicht realisierbar zu sein, da kein Hersteller bereit ist, Design-Unterlagen der Bausteine weiterzugeben.

Berücksichtigt man weiterhin die zunehmende Komplexität beim Übergang von Einzelbausteinen zum Rechner auf einer Platine, vom Einzelmikrocomputer zum Mehrrechnersystem, so bleibt für den Anwender nur der Weg: Test und Diagnose muß bereits beim Entwurf der Systeme berücksichtigt werden [2, 3]. Ein besonderes Beispiel ist der Entwurf der Mehrrechnerstruktur M3R [2].

Welche Testmöglichkeiten bieten sich an? Verschiedene Verfahren werden beschrieben als

- Testverfahren während der Chip-Herstellung,
- Prüfung fertiger Bausteine [1].

In [1] werden auch detailliert Testmethoden für Mikroprozessoren angegeben. In [4] werden verschiedene Verfahren aufgezählt:

- Simulation der Logik,
- Vergleichsmethode,
- Zufallsmethode.

Diese Tests umfassen hauptsächlich die Prüfung einzelner hochintegrierter Bausteine und nicht den Test von Systemen. Ein Weg in Richtung des Systemtests liegt im funktionellen Leiterplattentest [5, 6, 7], der hauptsächlich im Prüffeld Verwendung findet.

Im folgenden wird auf Methoden eingegangen, die bei der Entwicklung von Mikrocomputersystemen, wie bei dem in dieser Serie beschriebenen Arbeitsplatzcomputer, Verwendung gefunden haben.

2 Hilfsmittel zur Entwicklung von Mikrocomputersystemen

Für einfache Mikrocomputeranwendungen bieten sich problemlose, leicht zu bedienende Testgeräte an [8, 9], die darüber hinaus auch den Vorteil haben, für Lern- und Schulungszwecke eingesetzt werden zu können. Bei größeren und damit komplizierteren Systemen muß man jedoch umfassendere Testmöglichkeiten einsetzen. Dazu kennt man heute zwei Kategorien von Verfahren:

- Logikanalysatoren,
- In-Circuit-Emulatoren.

2.1 Logikanalyse

Logikanalyse bedeutet die Auswertung digitaler Informationen zum Zwecke der Überprüfung ihrer Richtigkeit. Ein Hilfsmittel hierfür ist der Logikanalysator, der durch den verstärkten Einsatz der Mikroprozesso-

ren seinen Siegeszug angetreten hat. Der Einsatz dieses Werkzeugs ist von Mitarbeitern der bekanntesten Hersteller in |10, 11, 12| beschrieben worden. Das Grundprinzip des Logikanalysators liegt in der parallelen Aufzeichnung (Speicherung) von Informationen (statischer Art: Daten, dynamischer Art: Störimpulsen), wenn bestimmte Suchbedingungen gegeben sind (Trigger). Das Gerät kann dabei sowohl für die Programmentwicklung, wie auch für die Hardwareentwicklung eingesetzt werden.

2.2 In-Circuit-Emulation

Auf einer höheren logischen Stufe anzusiedeln ist die Methode der In-Circuit-Emulation |13|. Sie unterstützt vor allem den Programmentwickler. In dieser Phase wird in einem Entwicklungssystem (vollständiges Mikroprozessorsystem, z. B. Siemens-Mikrocomputer-Entwicklungssystem SME 800 |14|) ein Programm erstellt (Texteditor), übersetzt (Compiler) und getestet (Emulator). Dieser Vorgang ist streng genommen der einer Simulation, wie sie auf Host-Rechnern durchgeführt wird.

Im nächsten Schritt wird dann das zu entwickelnde Prototypensystem mit dem Entwicklungssystem über das In-Circuit-Emulationssystem (z. B. Emulation- und Testadapter ETA80 |15|) verbunden (*Bild 1*). Dazu wird an Stelle des Mikroprozessors des Prototypensystems das Emulatorsystem gesteckt. Entwicklungssystem und Prototypensystem sind dann über eine „Nabelschnur" miteinander verbunden. Dabei wird die im ersten Schritt entwickelte Software zunächst noch im Entwicklungssystem ablaufen. Erst dann, wenn keine schwerwiegenden Hardwarefehler im Prototypensystem festgestellt werden („fatal errors"), wird die Software segmentweise (Datenbereich, Programmbereich, Stackbereich) in die Speicher des Prototypensystems ausgelagert.

Anschließend wird die Peripherie getestet. Schließlich wird die „Nabelschnur" entfernt, der vorher entfernte Prozessor ins Prototypensystem gesteckt und ein erster Lauf unter realen Verhältnissen versucht.

2.3 Kombination beider Verfahren

In sehr vielen Fällen wird man mit den beschriebenen Methoden – getrennt eingesetzt – Erfolg haben. In kritischen Fällen muß man sie kombinieren. Man spricht dann vom „hardware trace" |13|. Der Logikanalysator gibt dabei Auskunft über Zeitverhalten. Er speichert Bussignale und die Zyklen einzelner Befehle.

In besonders zeitkritischen Fällen muß man ab einer gewissen Stufe der Entwicklung auf die Emulation ganz verzichten, da sie nie unter Echtzeitbedingungen ablaufen kann.

3 Weitere Hilfsmittel für die Entwicklung des Arbeitsplatzcomputers

Für den Aufbau des Arbeitsplatzcomputers wurden zusätzliche Methoden entwickelt, die die Hardware sowohl auf der Einzelrechnerebene testen, wie auch den Verbundbetrieb dieser Mehrrechnerstruktur (siehe Teil 1 und 2 dieser Aufsatzreihe).

3.1 Einzelrechner

Zum Test der Einzelrechner (CPU, Speicher, Peripherie) wurde ein *Test-* und *Diagnoseadapter* für die Bussignale entwickelt (*Bild 2*). Er wird über eine Entkopplungsschaltung an den Bus des Mikrocomputers angeschlossen. Die nachgeschaltete Elektronik wertet die Bussignale aus und erzeugt entsprechende Steuersignale für die Anzeigen. Damit können alle Bussignale dauernd beobachtet werden. Im Fehlerfall läßt sich dann schnell feststellen, mit welchen Hilfsmitteln und in welcher Richtung zur Fehlereingrenzung weiter gearbeitet werden muß.

Über unterschiedlich farbige Leuchtdioden werden für jedes Steuersignal zwei Zustände angezeigt: Der momentane und der vorhergehende Zustand. Die Hexadezimal-Anzeigen des Daten- und Adressenbusses werden mit den Zugriffssignalen getriggert. Bleibt der Mikrocomputer stehen, kann man mit Hilfe des Adapters folgende Fragen rasch beantworten: Welcher Art war das letzte Steuersignal? Unter welcher Adresse wurde zugegriffen? Welches Datum war dieser Adresse zugeordnet? Über Schalter kann man wählen, welches oder welche Kombination von Bussignalen zur Triggerung herangezogen werden (dies gilt auch für beliebige Kombinationen der Daten- oder Adresseninformationen).

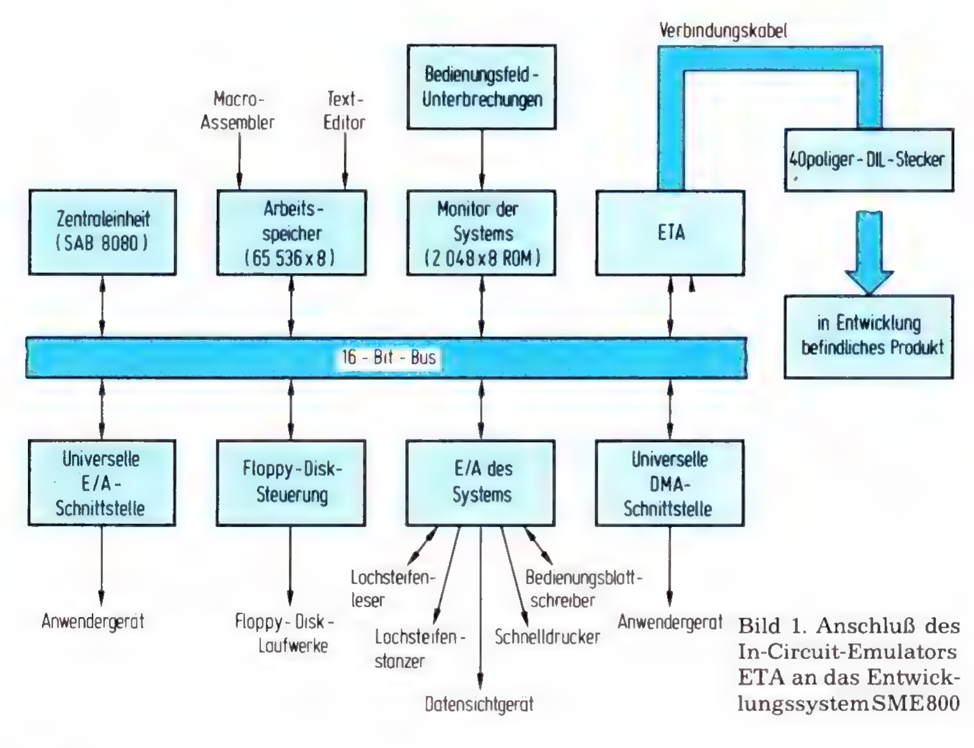


Bild 1. Anschluß des In-Circuit-Emulators ETA an das Entwicklungssystem SME 800

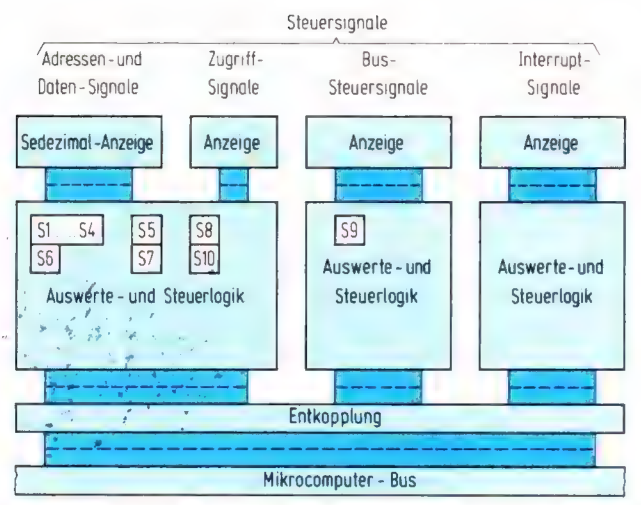


Bild 2. Blockschaltung des Diagnose- und Testadapters

Jedes Bus-Steuersignal wird mit einer nachtriggerbaren monostabilen Kippstufe verlängert, damit auch kurze Einzelreaktionen erkannt werden können. Darüber hinaus können Haltepunkte gesetzt werden, so daß in kritischen Fällen auch im Single-Step-Verfahren gearbeitet werden kann.

Auf dem Einzelrechner laufen *Testprogramme für die Einzelfunktionen* des Rechners ab:

– CPU-Test
– Speichertestroutinen
– Peripherietests

Dabei wird auf das Selbsttestverfahren von Mikrorechnern zurückgegriffen [16, 17]. Damit lassen sich beim Urladen des Systems nach Einschalten der Stromversorgung und in regelmäßigen Zeitabständen beim späteren Systemlauf Aussagen über den Funktionszustand des einzelnen Mikrocomputers machen.

Ein wichtiges Hilfsmittel ist die *Spannungsüberwachung* von Mehrspannungsnetzgeräten für die Mikrocomputer (*Bild 3*). Erfahrungsgemäß denkt man an das fehlerhafte Arbeiten des Netzteils bei der Fehlersuche immer erst zuletzt, meist wenn man mit der logischen Fehlersuche nicht mehr weiterkommt. Dazu ist es sinnvoll, das Überschreiten bzw. Unterschreiten von Toleranzwerten der einzelnen Quellen mit Leuchtdioden anzuzeigen, um so in Abhängigkeit von einer Referenzspannung den augenblicklichen Zustand ablesen zu können.

Als sinnvoll hat sich auch der *Anschluß einer Diagnosekonsole* erwiesen, über die Zugriff zu den auf Lesespeichern (ROMs oder PROMs) vorliegenden Testprogrammen möglich ist. Diese Konsole erlaubt auch die Steuerung des Testablaufs durch den Entwickler (beim späteren Einsatz: durch den Benutzer) abhängig vom Testfortschritt, der am Bildschirm angezeigt werden kann. Im einfachsten Fall wird dazu die Ein-/Ausgabeeinrichtung verwendet, über die der Benutzer normalerweise mit dem System verkehrt.

3.2 Mehrrechnerstruktur

Die einzelnen Mikrocomputer werden, wie in Teil 1 und 2 dieser Reihe beschrieben, über ein Kommunikationssystem (Bus) miteinander verbunden. Für den Test des Gesamtsystems werden sämtliche Einzeltestverfahren, wie vorstehend beschrieben, benutzt. Darüber hinaus wurden noch weitere Möglichkeiten geschaffen:

Auf den *Bussteuerbaugruppen jedes Rechners* sind *Anzeigen* vorgesehen, um über den aktuellen Zustand des Kommunikationssystems jeder Zeit informiert zu sein. Damit wird der Kommunikationszustand jedes Einzelrechners angezeigt. Darüber hinaus gibt es Eingriffsmöglichkeiten auf den einzelnen Rechner, die es ermöglichen, einen defekten Rechner vom übrigen System zu isolieren und abzuschalten (für Wartungszwecke) bzw. wieder dazuzuschalten.

In jedem Rechner liegen auf PROMs *Testprogramme für die Kommunikation* vor, die vor allem den Test zeitkritischer Abläufe ermöglichen. Um ein gezieltes Starten dieser Programme in allen Rechnern gleichzeitig zu initialisieren, wurde eine *Zusatzschaltung* entwickelt, die es erlaubt, den Ausgang der Diagnosekonsole parallel auf alle Diagnoseeingänge der einzelnen Rechner zu schalten und die Rechner zur Aufnahme dieser Information zu unterbrechen. Darüber hinaus kann der Diagnoseausgang eines beliebigen Rechners (schaltbar) angezeigt werden.

4 Testhilfen während des Systemlaufs

Für den Test der Einzelkomponenten einer Mehrrechnerstruktur und des gesamten Systems sind verschiedene Strategien denkbar. Hier wurde die Methode gewählt, einen Funktionsrechner für Diagnose und Test entweder temporär (zu den anderen Aufgaben) oder ausschließlich zu benutzen. Über diesen Rechner werden beim Einschalten des Systems (Urla-

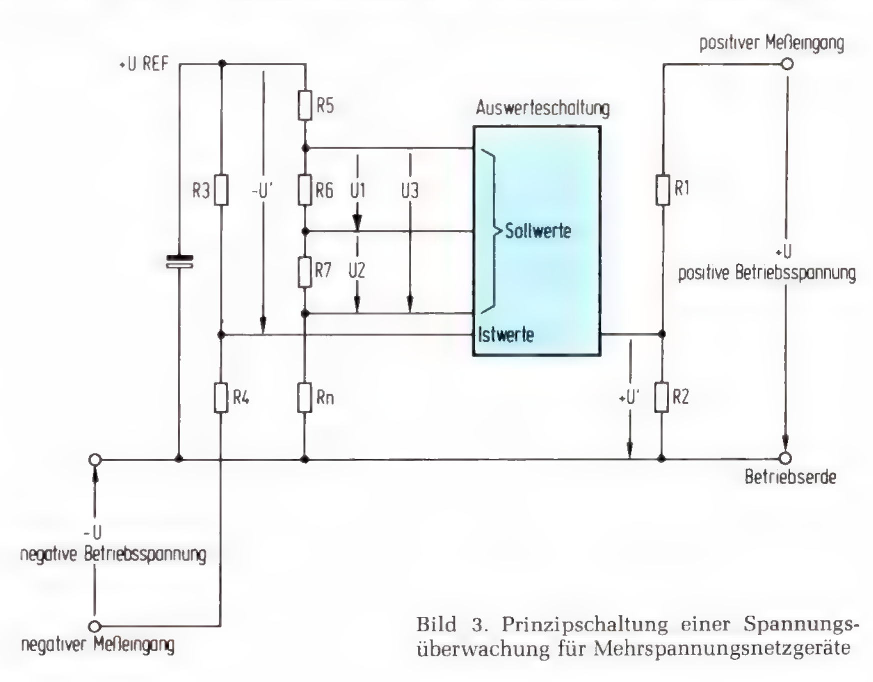


Bild 3. Prinzipschaltung einer Spannungsüberwachung für Mehrspannungsnetzgeräte

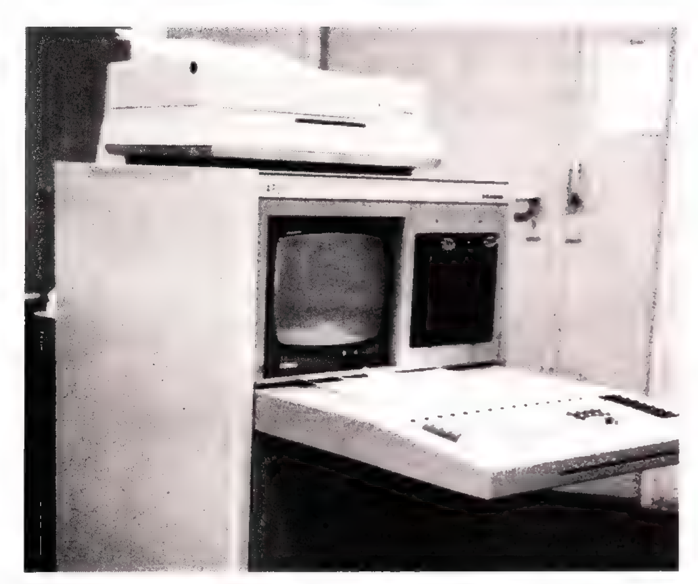


Bild 4. Prototyp des Arbeitsplatzcomputers (Laboraufbau)

den) und in bestimmten Zeitabständen (einstellbar über Timer) Testprogramme auf den einzelnen Komponenten des Systems initialisiert, die Ergebnisse gesammelt und ausgewertet.

Werden fehlerhafte Ergebnisse ermittelt, so erfolgt eine weitergehende Prüfung des Rechners, die nach einer Testwiederholung (um Zufallsfehler auszuschließen) und Fehlerwiederholung zur Abschaltung führt. Das übrige System arbeitet weiter. Die Aufgaben des abgeschalteten Rechners werden nach einer möglichen Neukonfiguration von den anderen übernommen. Der abgeschaltete Rechner wird angezeigt und kann nun vom Benutzer (Wartungstechniker) für Reparaturmaßnahmen ausgewechselt werden.

Eine ausgetauschte Baugruppe wird durch den Diagnoserechner getestet, ehe eine Neukonfiguration des Systems mit dem ursprünglichen Ausbau durchgeführt wird.

5 Konstruktive Maßnahmen

In den vorangegangenen Teilen dieser Serie wurden generelle und detaillierte Aussagen über Prinzip, Sinn, Ablauf von Prozessen einer Mehrrechnerstruktur und einige Anwendungen gemacht. Der Nachweis der Gültigkeit läßt sich jedoch erst durch den Aufbau eines Prototypensystems erbringen (*Bild* 4). Gewählt wurde die Form eines Kompaktgerätes, in dem alle Teile untergebracht sind und das – im Unterschied zum Tischmodell – keine dauernde Standfläche benötigt, sondern bei Bedarf an den Schreibtisch des Bearbeiters geschoben werden kann.

Hier schließt sich der Bogen zu der eingangs erwähnten Forderung: Test und Diagnose muß bereits beim Entwurf eines Systems berücksichtigt werden. *Bild 5a* zeigt den Rechnerblock von der Vorderseite. Man erkennt als Baugruppenträger sogenannte „card cages", die jeweils einen Mikrocomputer darstellen (im Bedarfsfall können mehrere „card cages" einen Rechner bilden). Es wurden Standardplatinen des Systems SBC 8020 und 8612 mit den entsprechenden Erweiterungsplatinen verwendet. Man sieht ebenfalls die Bussteuerplatinen in jedem Rechner mit den Anzeige-LEDs. *Bild 5b* zeigt den Rechnerblock von der Rückseite. Hier werden die beiden Bussysteme sichtbar:

- der interne Bus der einzelnen Mikrocomputer, der Bestandteil der Baugruppenträger ist,
- der Kommunikationsbus für den Verkehr der Rechner untereinander (steckbare Leiterplatten im unteren Teil des Bildes).

Der Kommunikationsbus wurde im Sinne der Fehlererkennung so bemessen, daß hier nicht mit TTL-Pegel (0...5 V), sondern mit symmetrischen Signalen und hohem statischen Störabstand gearbeitet wird ($\pm$12 V). Dafür wurden Treiber- und Empfängerbausteine von Texas Instruments (SN 75150 und 75154) verwendet. Außerdem wird auf dem Bus das Verfahren der Paritätssicherung eingesetzt.

Schließlich wird im Sinne der Modularität der Einzelkomponenten des Systems keine gemeinsame Stromversorgung verwendet. D. h. jeder Rechner hat sein eigenes Netzgerät, das oberhalb der „card cages" angeordnet ist (s. Bild 5). Der Anschluß an die 220-V-Versorgung geschieht nur einmal, so daß hier von einem dritten Bus zu sprechen ist.

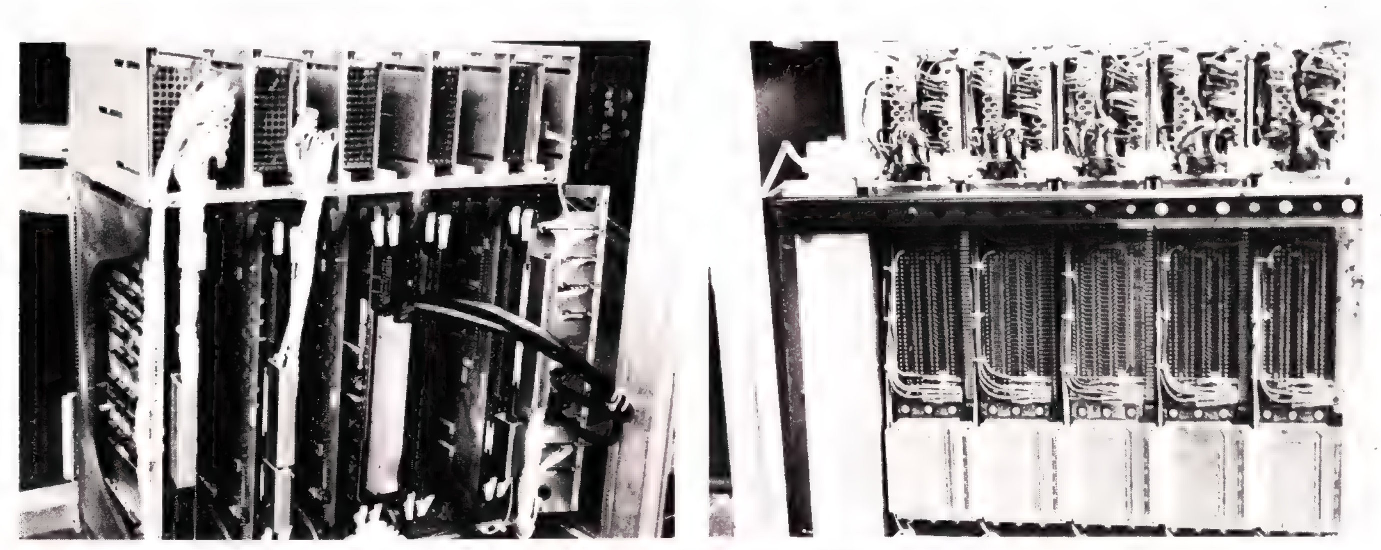
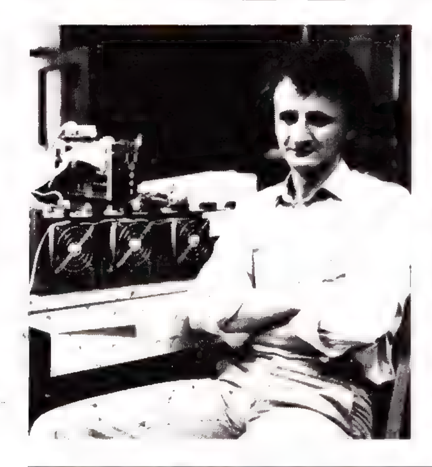


Bild 5. Rechnerblock des Arbeitsplatzcomputers, links Vorderansicht, rechts Rückseite



Dipl.-Ing. Eduard Scheiterer ist geboren in Wien. Er studierte Nachrichtentechnik an der Techn. Universität Wien. 1978 trat er in die Siemens AG in München ein und beschäftigt sich mit Aufbau und Entwicklung von Mehrrechnersystemen.
Hobbys: Klassische Musik, Segeln
Telefon: (089) 6782-4239
ELEKTRONIK-Leser seit 1978

Dipl.-Ing. Anton Sauer und *Ing. (grad.) Hans Thinschmidt* wurden bereits im 1. bzw. 2. Teil vorgestellt.

6 Schlußbemerkungen

Es hat sich gezeigt, daß die Maßnahmen für Test und Diagnose, die bereits beim Entwurf dieses Systems getroffen wurden, einen erheblichen Einfluß auf die Stabilität und Sicherheit des Systems im praktischen Einsatz hatten. Man ist sich bewußt, daß damit keine endgültigen Ergebnisse erzielt werden konnten. Das Problem des Tests von Mehrrechnerstrukturen wird weiter untersucht. Hoffnungsvolle Ansätze in Richtung verteilter In-Circuit-Emulationssysteme bieten sich an. Man hofft, darüber in einem weiteren Aufsatz berichten zu können.

Literatur

1 *Görke, W.*: Testmöglichkeiten für LSI-Schaltkreise. Elektronische Rechenanlagen, Bd. 20 (1978), H. 5, S. 220...227.
2 *Nilsson, S. A.*: M3R – Ein modulares Mehrmikrorechner-System mit Restverfügbarkeit und Prozeßsicherungsstruktur. Elektronische Rechenanlagen, Bd. 20 (1978), H. 3, S. 115...123.
3 *Lowe, L.*: Designing for testability. Microprocessors and Microsystems, Bd. 3 (1979), H. 1, S. 3...6.
4 *Bollen, H.*: Wichtiger denn je – der Test von LSI-Bauelementen. ELEKTRONIK 1978, H. 13, S. 77...80.
5 *Pöhlmann, B., Giesbrecht, D.*: Schnelle Fehlerdiagnose an bestückten Leiterplatten. ELEKTRONIK 1979, H. 5, S. 53...56
6 *Scipio, R.*: Automatischer Digital-Tester arbeitet mit vier Prozessoren. ELEKTRONIK 1978, H. 8, S. 67...70, 76.
7 *Davis, H.*: Testing µP-boards can be easier – if you design your own tester. Electronic design, Bd. 27 (1979), H. 5, S. 196...198.
8 *Wesselhöft, U.*: Mikrocomputer-Daten und -Adressen schrittweise angezeigt. ELEKTRONIK 1977, H. 6, S. 103...104.
9 *Gorka, W.*: Einfaches Testgerät für ein 8080-System. ELEKTRONIK 1978, H. 10, S. 77...78.
10 *Baumann, T.*: Fehlersuche mit modernen Logikanalysatoren. ELEKTRONIK 1978, H. 6, S. 84...88.
11 *Quick, P.*: Praxis der Logikanalyse. ELEKTRONIK 1979, H. 7, S. 55...59.
12 *Lorentzen, R.*: Troubleshooting Microprocessors with a Logic Analyzer System. Computer Design, Bd. 18 (1979), H. 3, S. 160...164.
13 *Dum, S., Philpott, D.*: In-Circuit emulation aids microprocessor system design. Electronic Engineering 50 (1978) 615, S. 87, 89...90.
14 Einführung: Siemens Mikrocomputer-Entwicklungssystem. Siemens AG, B-1663 (mit weiteren Literaturangaben über dieses System)
15 Emulations- und Testadapter ETA80. Siemens AG, B-1773.
16 *Ebel, B.*: Mikroprozessor Selbsttest. Elektronische Rechenanlagen, Bd. 20 (1978), H. 4, S. 186...194.
17 *Nilsson, S. A.*: Selbsttestverfahren für Mikrorechner. VDI-Berichte 1978, Nr. 328, S. 77...85.
18 *Will, B.*: Erhöhung der Ausfallsicherheit bei Systemen verteilter Intelligenz. ELEKTRONIK 1979, H. 7, S. 69...73.

Ing. (grad.) Horst Huse

Multi-Mikrocomputer-System modular aufgebaut

Leistungsfähige 16-Bit-Mikroprozessoren erlauben die Realisierung kompakter Module für Mehrrechnersysteme mit direkter Kopplung über den Systembus. Im Gegensatz zu dezentralen, meist über serielle Bussysteme lose gekoppelten Mehrrechnersysteme [1] wird die Systemkopplung dazu verwendet, bereits innerhalb eines 19-Zoll-Einbaurahmens die Rechenleistung zu erhöhen. Anstelle einer kostspieligen und unüberschaubaren Zentralisierung werden Systeme mit einer funktionsorientierten Architektur angestrebt [2]. Dieses setzt voraus, daß der Systembus einen hohen Datendurchsatz erlaubt und Multiprozessorfähigkeit besitzt. Außerdem muß der mechanische Aufbau ausreichende Flexibilität bieten, wie z. B. Mischbarkeit von zwei Platinengrößen innerhalb eines Einbaurahmens. Diese Forderungen werden bei Verwendung des E-BUS als genormte Systemschnittstelle erfüllt [3]. Dieser Beitrag beschreibt Möglichkeiten, Grenzen und Vorteile der Systembuskopplung anhand eines praktischen Beispiels.

1 Warum Multi-Mikrocomputer-Systeme?

In den letzten Jahren wurde bei Mikroprozessoren eine beachtliche Leistungssteigerung durch Verbesserung der Architektur, Erweiterung des Befehlssatzes und Einsatz kleiner Halbleitergeometrien (z. B. SMOS – „Scaled MOS") erzielt. Oft kann die höhere Geschwindigkeit (z. B. Speicherzykluszeiten von < 200 ns) auf der Systemebene nicht wirtschaftlich genutzt werden. Der Datendurchsatz wird durch die Zugriffszeit der Speicher, die Durchlaufverzögerungszeit der Treiber und die Geschwindigkeit der Ein-/Ausgabe begrenzt. Wird ein zentrales Mikrocomputersystem z. B. unter Verwendung von ECL-Logik oder eines 32-Bit-Rechners für einen höheren Datendurchsatz konzipiert, so steigen die Kosten überproportional an. Es ist aus diesem Grunde oft wirtschaftlicher, in industriellen Systemen noch höhere Systemleistungen durch den Einsatz mehrerer kostengünstiger 8- oder 16-Bit-Mikroprozessoren anzustreben. Möglichkeit dazu bietet die Verwendung von Mikroprozessoren und Baugruppen mit komplexeren Funktionen, wie z. B. Kommunikations- und Ein-/Ausgabesteuerungen. Durch die Intelligenz der Ein-/Ausgabemodule lassen sich Teile der Software des zentralen Mikroprozessors (Master) in den Peripherie-Mikrocomputer (Slave) verlagern. Aus einem zentralisierten Mikrocomputersystem wird ein dezentralisiertes Multi-Mikrocomputersystem. So kann z. B. in der Software des Peripherie-Mikrocomputers ein HDCL-Übertragungsprotokoll oder Floppy-Disk-Dateiorganisation realisiert werden.

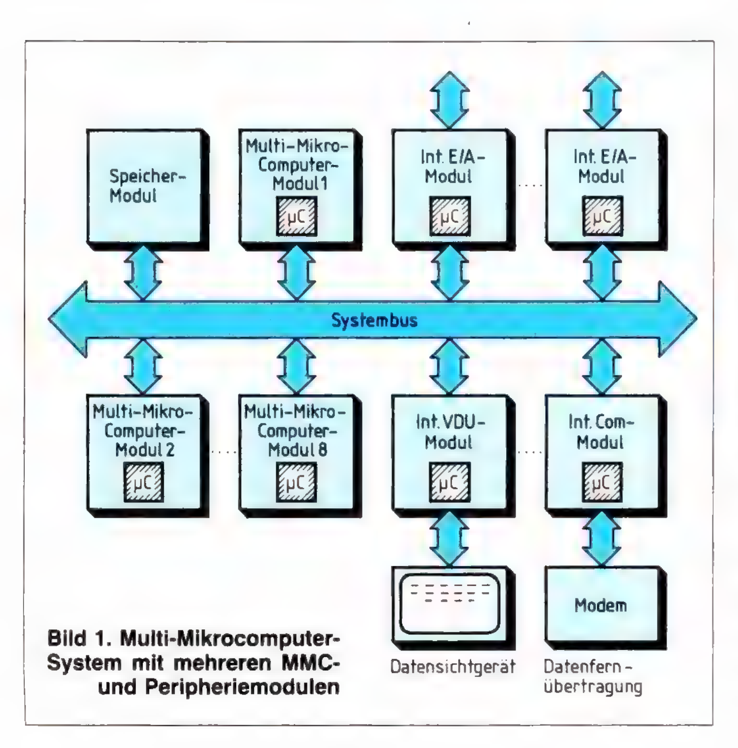


Bild 1. Multi-Mikrocomputer-System mit mehreren MMC- und Peripheriemodulen

Bei industriellen Anwendungen lassen sich auch zeitkritische Aufgaben wie Regelkreise oder Positionierungen in einen „intelligenten" Peripheriemodul verlagern. Voraussetzung ist jeweils, daß der Peripherie-Mikrocomputer mit dem Master kommunizieren kann und Zugriff zu den notwendigen Ein-/Ausgabeinformationen hat.

Bei größeren Systemen werden bestimmte Funktionen (Prozesse) oder ganze Programmodule (Tasks) auf mehrere Multi-Mikrocomputermodule (MMC-Module) verteilt. Sind in einem System komplexe und zeitkritische Berechnungen notwendig, wie z. B. dreidimensionale Bahnberechnungen, sollte diese Aufgabe einem einzelnen Modul übertragen werden.

Die Verteilung der Gesamtfunktion des Systems auf parallel arbeitende Multi- oder Peripherie-Mikrocomputermodule (*Bild 1*) bietet außer der höheren Systemleistung noch folgende Vorteile:

- Entlastung des zentralen Mikrocomputers von zeitkritischen Abläufen.
- Aufteilung eines großen zentralen Softwarepaketes in mehrere kleinere und damit wartungsfreundlichere Programme.
- Vermeidung des überproportional steigenden Speicher- und Programmverwaltungsaufwandes von großen zentralisierten Systemen.
- Einfachere Überwachungsmöglichkeiten durch parallel arbeitende Mikrocomputer während des Betriebes.

Speziell die steigenden Anforderungen an die Fehlersicherheit industrieller Systeme lassen sich mit Multi-Mikrocomputerstrukturen wirtschaftlich vertretbar realisieren, da gezielt Redundanz eingeplant werden kann.

2 Kopplung der Mikrocomputer-Hardware

Dezentralisierte Mikrocomputermodule werden entweder für eine bestimmte Funktion, z. B. Datenübertragung oder Lageregelung, oder als universell einsetzbare Multi-Mikrocomputermodule ausgelegt. In beiden Fällen muß sichergestellt sein, daß der Mikrocomputer Zugriff zu den notwendigen Ein-/Ausgabeinformationen hat, und mit dem Master kommunizieren kann. Die Wahl des Kommunikationsweges zum Master ist davon abhängig, ob ein direkter Zugriff zu Speicher- oder Ein-/Ausgabemodulen am Systembus erforderlich ist oder nicht.

Ist dieses nicht der Fall, d. h. der erforderliche Speicher und die Ein-/Ausgabeverbindungen befinden sich auf der Platine, kann die Kommunikation zum Master durch eine lose Kopplung beider Bussysteme realisiert werden. Ist ein Zugriff auf Module am Systembus notwendig, muß eine enge Buskopplung vorgesehen werden. Diese Art der Kopplung ist auch dann erforderlich, wenn der Slave-Modul bei Ausfall des Masters dessen Aufgaben teilweise übernehmen soll.

2.1 Lose Buskopplung

Die lose Buskopplung zwischen zwei Bussystemen dient zur Kommunikation zwischen Master- und Slave-Mikrocomputermodulen und wird mittels einer Koppeleinheit realisiert (*Bild 2*). Art und Größe der Koppeleinheit ergibt sich aus der Datenmenge und der Geschwindigkeit, mit der die Kommunikation erfolgen soll.

Bei der einfachsten Art der Kommunikation wird ein Slave-Modul durch einen Interrupt vom Master zur Ausführung einer definierten Funktion veranlaßt. Der Slave meldet die korrekte Ausführung wiederum durch einen Interrupt zum Master. Sollen Parameter oder ganze Datenblöcke zwischen beiden Modulen ausgetauscht werden, müssen die Kommunikationsmöglichkeiten erweitert werden.

Es stehen dem Systementwickler drei weitere Möglichkeiten zur Auswahl:

- Kopplung über Speicherworte,
- Ein-/Ausgabeports oder den
- DMA-Speichertransfer.

Außer bei der DMA-Kopplung wird der Systembus durch Zugriffe des Slave-Moduls auf die Koppeleinheit nicht belastet. Bei allen drei Kopplungsarten ist auch eine „Multiprozessorfähigkeit" des Systembusses nicht zwingend erforderlich. Aus diesen Merkmalen resultieren die wichtigsten Eigenschaften der losen Buskopplung.

2.1.1 Kommunikation über Speicherworte

Für die Kommunikation über Speicherworte wird ein spezieller Speicherbereich so ausgelegt, daß er inner-

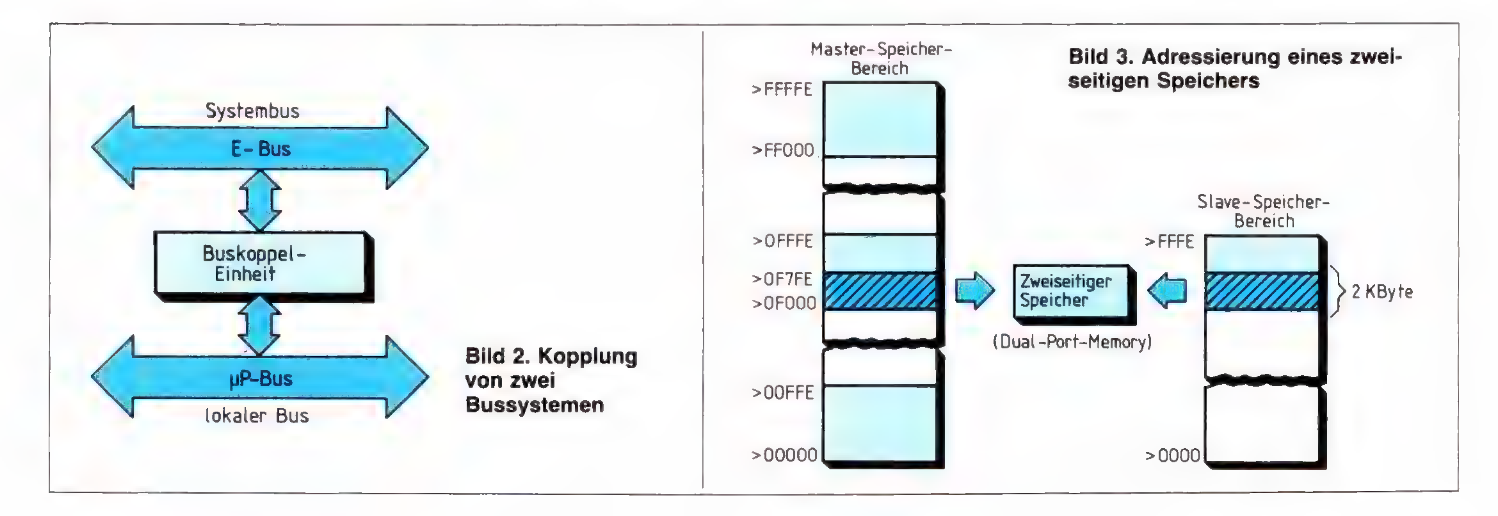


Bild 2. Kopplung von zwei Bussystemen

Bild 3. Adressierung eines zweiseitigen Speichers

halb des Adreßbereichs des Master- und Slave-Moduls liegt. Es handelt sich somit um einen Bereich, der von zwei Seiten adressierbar ist, und wird deshalb als zweiseitiger Speicher oder „Dual-Port-Memory" (DPM) bezeichnet. Das Beispiel in *Bild* 3 zeigt einen 2 KByte großen Speicherblock, welcher innerhalb des 1 MByte-Adreßbereiches des Masters und des 64-KByte-Bereiches des Slaves liegt.

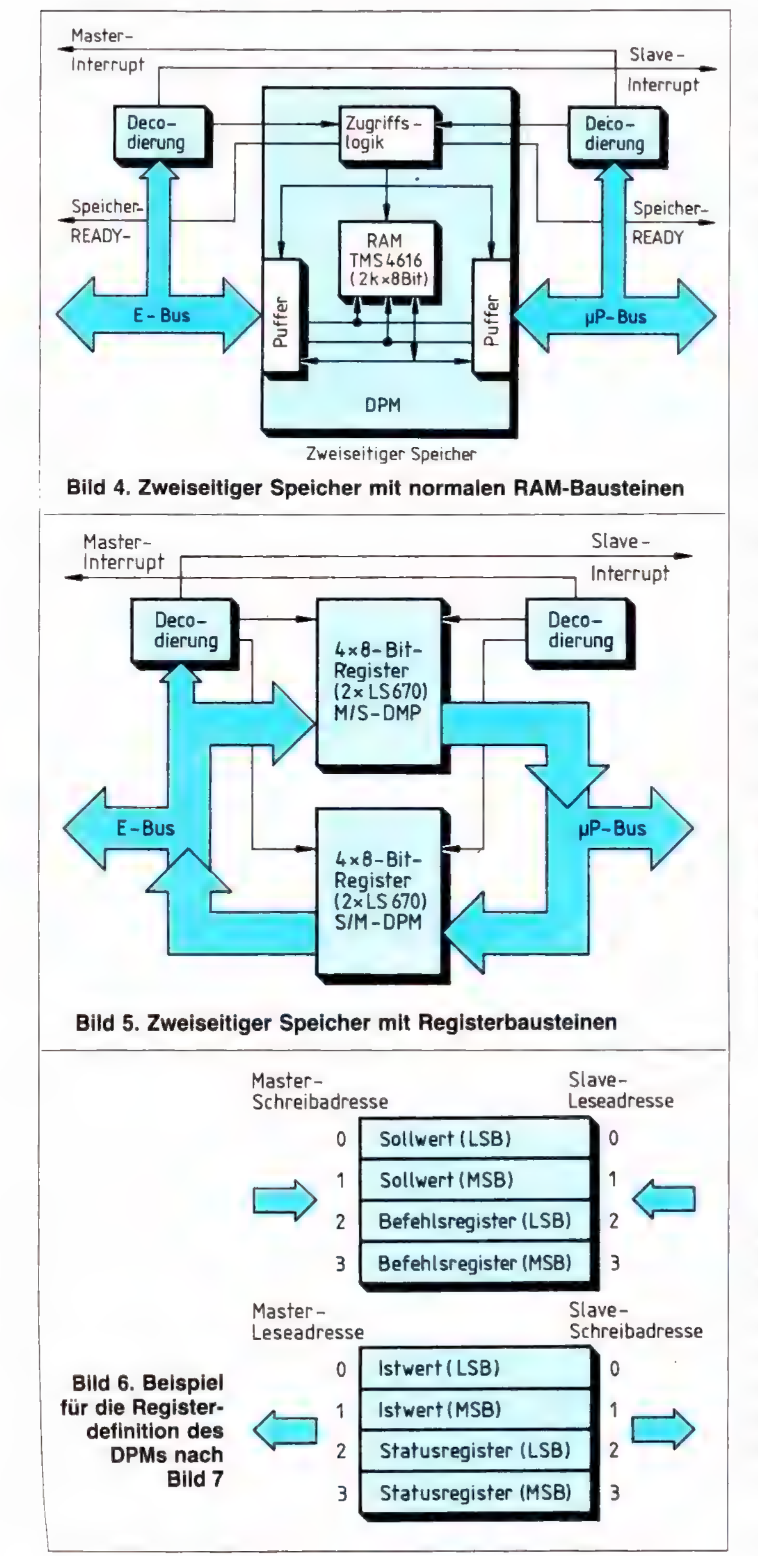


Bild 4. Zweiseitiger Speicher mit normalen RAM-Bausteinen

Bild 5. Zweiseitiger Speicher mit Registerbausteinen

Bild 6. Beispiel für die Registerdefinition des DPMs nach Bild 7

Die absoluten Adressen für beide Seiten des DPMs sind frei wählbar. Werden normale RAM-Speicherbausteine, z. B. der Typ TMS 4016 (2 K × 8 Bit organisiert), für das DPM verwendet, so kann nicht gleichzeitig von zwei Seiten gelesen bzw. geschrieben werden. Durch eine Zugriffsteuerung wird aus dem einfachen Speicher ein Pseudo-DPM (*Bild* 4). Die aus der Adreßdecodierung generierten Speicheranforderungen werden von der Zugriffslogik ausgewertet. Ist der Zugriff von nur einer Seite gewünscht, so erhält diese sofort eine Zuteilung. Treten beide Anforderungen gleichzeitig auf, hat der Systembuszugriff Vorrang. Der Speicherzyklus des Slaves oder Masters kann über die Steuerung der READY-Signale verzögert werden. Das Pseudo-DPM führt in der Regel zu keiner nennenswerten Erhöhung der Systembusbelegung. Die Festlegung der Funktion bzw. die Bedeutung einzelner Speicherworte des DPMs wird an Hand der Kommunikationsanforderungen festgelegt. Speicherstellen die für den Master als Schreib-/Lese-RAM definiert sind, dürfen vom Slave nur gelesen werden. Dies gilt umgekehrt ebenso für den Master. Eine interruptgesteuerte Kommunikation ist dadurch realisierbar, daß beim Schreiben auf eine bestimmte Speicheradresse automatisch ein Interrupt für den Master bzw. den Slave erzeugt wird.

Ist die Menge der auszutauschenden Daten gering, so ist ein kleiner Speicher mit schnellen Registerbausteinen, wie den Typen SN74SL170 bzw. 'LS670, ausreichend. Da diese Bausteine gleichzeitiges Lesen und Schreiben erlauben, kann auf eine Zugriffslogik verzichtet werden. *Bild* 5 zeigt das Blockdiagramm eines DPMs mit je 4 Registern zu 8 Bit für die Kommunikation Master ⇒ Slave und Slave ⇒ Master. Der sequentielle Transport mehrerer Datenbytes über ein Register wird durch eine einfache Handshake-Prozedur per Interrupts oder Softwareflags realisiert. Ein Beispiel der Registerdefinition des DPMs für einen einfachen Regelkreis (Drehzahl- oder Positionssteuerung) zeigt *Bild* 6. Es ist zu beachten, daß bei der Realisierung des DPMs nach Bild 5 geschriebene Wörter nicht wie beim Pseudo-DPM wieder auslesbar sind. Das Register-DPM zeichnet sich durch kurze Zugriffszeit (< 50 ns) und geringen Aufwand in der Hardware aus.

2.1.2 Kommunikation über Ein-/Ausgabeports

Die Kommunikation zu Slave-Mikrocomputern ist ebenso über Ein-/Ausgabeports realisierbar. Analog zum DPM muß ein zweiseitiges E/A-Port realisiert werden. Dieses liegt wiederum im E/A-Adreßbereich des Masters und Slaves (Bild 3). Sind die Anforderungen an Datenmenge und Geschwindigkeit gering, läßt sich diese Art der Buskopplung am einfachsten aufbauen. Der Datenaustausch wird über Handshake-E/A-Ports und/oder Interrupts gesteuert, um den Leistungs- und Hardwareaufwand gering zu halten.

Einfache intelligente Peripheriesteuerungen mit Einchip-Mikrocomputern (µC) können so effektiv an den

Systembus gekoppelt werden (*Bild 9*). Im Gegensatz zur DPM-Kopplung gehen keine Speicheradressen verloren, wenn der Master-Mikrocomputer über einen separaten Ein-/Ausgabeadreßbereich verfügt. Ist der E/A-Bereich am Systembus und µP-Bus groß genug, so kann auch eine Kopplung mittels DPM, adressiert über E/A-Adressen, vorgenommen werden.

2.1.3 Datentransfer per DMA

Peripherie-Mikrocomputermodule, die größere Datenblöcke mit hoher Geschwindigkeit transportieren müssen, sollten über einen direkten Systemspeicherzugriff verfügen. Dieser schnelle DMA-Kanal wird direkt vom Master oder, an Hand der im DPM übergebenen Parameter, vom Slave initialisiert. Die sich aus dem DMA-Transfer ergebende zeitliche Belastung des Systembus ist geringer als die eines programmierten Datentransfers des Masters vom DPM zum Systemspeicher. Ist der Systembus nicht multiprozessorfähig, wird der Master- und Slave-Mikrocomputer während des DMA-Transfers in den HOLD-Mode gesteuert. Für den multiprozessorfähigen E-BUS stellt eine DMA-Steuereinheit einen Busmaster dar und muß deshalb über einen Buszuteiler (Arbiter) verfügen. Das typische Blockdiagramm eines Slave-Mikrocomputermoduls mit DMA-Kanal zeigt *Bild 8*. Peripheriegerätesteuerungen, wie z. B. Floppy- oder Hard-Disk, Grafik-Display oder schnelle Achsensteuerungen, sind mit dieser Konfiguration effektiv zu realisieren. Der Slave-Mikrocomputer muß über möglichst viel lokalen Programm- und Datenspeicher verfügen, so daß auch komplexere Programme vom Master in den Slave verlagert werden können. Besteht eine Softwarekompatibilität zwischen beiden, ist eine Übertragung der Software ohne größere Änderungen zu realisieren.

2.1.4 Lose Kopplung in großen Systemen

Ist eine größere Anzahl von Ein-/Ausgabeverbindungen notwendig, und sind diese nicht auf einem Modul herzustellen, ist eine Erweiterung des lokalen µP-Bus möglich. Sie kann auf mehrere Arten realisiert werden. Die einfachste Methode sind sogenannte „Piggy-Packs“; kleine Module, die direkt auf das Mikrocomputermodul gesteckt werden und eine begrenzte Speicher- oder Ein-/Ausgabeerweiterung zulassen.

Für industrielle Ein-/Ausgabeeinheiten ist diese Art jedoch nicht einsetzbar, da Steckverbinder und Kriechstrecken nicht den VDE-Bestimmungen entsprechen. Standard-Ein-/Ausgabemodule, welche auf den lokalen µP-Bus oder einen Systembus ausgelegt sind, erfüllen diese Anforderungen. In E-Systemen [4] ist deshalb bei

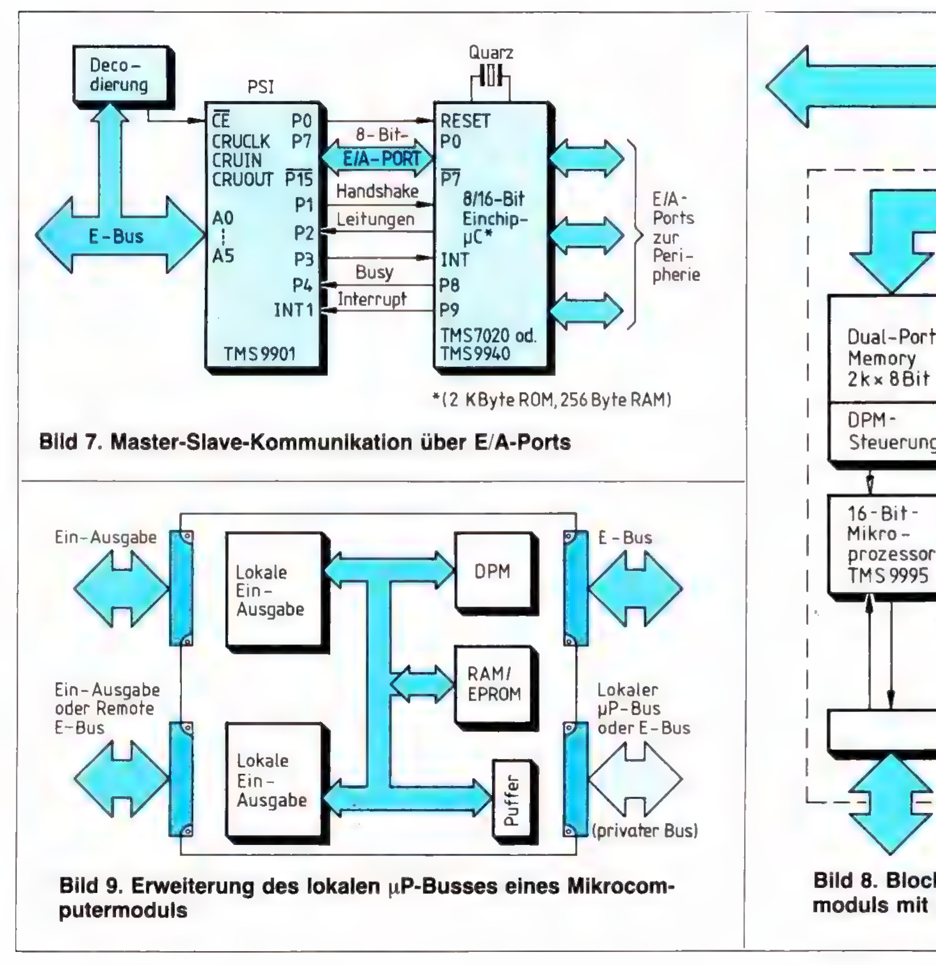


Bild 7. Master-Slave-Kommunikation über E/A-Ports

Bild 9. Erweiterung des lokalen µP-Busses eines Mikrocomputermoduls

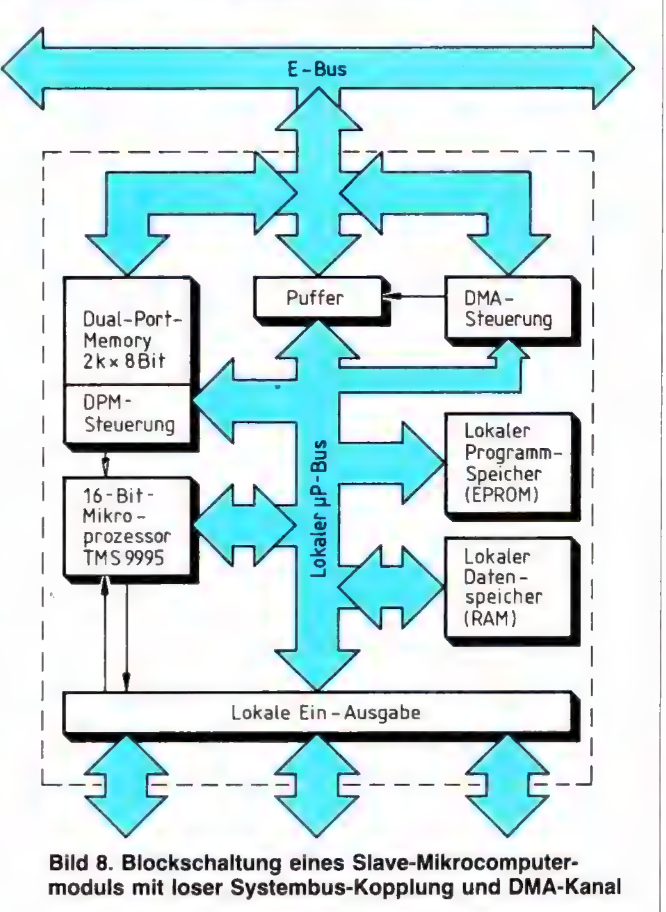


Bild 8. Blockschaltung eines Slave-Mikrocomputermoduls mit loser Systembus-Kopplung und DMA-Kanal

Verwendung des Doppel-Europakartenformates (230 × 160 mm²) der zweite Busstecker für einen lokalen µP-Bus oder einen zweiten E-BUS vorgesehen (*Bild 9*). Der zweite E-BUS ist, bei Verwendung einer Untermenge von Signalen, als einfacher E/A-Bus ausführbar. Die Untermenge für die industrielle bitserielle Ein-/Ausgabe (CRU) umfaßt nur 11 Leitungen. Bei geeigneter Pufferung ist der Anschluß von „Remote-" Ein-/Ausgabesteckrahmen mit einer Entfernung von bis zu 30 m, möglich. Ist die Anzahl der Einschübe innerhalb eines 19-Zoll-Steckrahmen für ein größeres Multi-Mikrocomputersystem nicht ausreichend, so ist in den meisten Anwendungen eine lose Kopplung mehrerer Steckrahmen sinnvoller, als eine „Verlängerung" des Systembusses. Bei einer Verlängerung wird der Datendurchsatz des Systembus durch zusätzliche Pufferung und größere Leistungslängen verringert. Die Leistungsgrenze des Systems wird herabgesetzt. Steht dagegen pro Steckrahmen ein Systembus mit mindestens einem µC-Modul zur Verfügung, wird der Datendurchsatz des Gesamtsystems durch Parallelverarbeitung erhöht. Innerhalb eines Steckrahmens können wiederum mehrere Mikrocomputer an einem Systembus arbeiten.

Die Kommunikation der Teilsysteme untereinander wird über „Message-Channels" vorgenommen. Wie schon bei der losen Kopplung von Peripherie-Mikrocomputermodulen bestimmt die auszutauschende Datenmenge die Art der Kopplung. Beschränkt sich die Datenmenge auf einige Parameter, Status- und Steuerinformationen, reicht eine Registerkopplung aus (Bild 5). Ist eine blockorientierte Kommunikation erforderlich, bietet sich ein Multi-Port-Memory (MPM) an. Der Hardwareaufwand läßt sich durch den Einsatz eines Pseudo-MPMs auf ein akzeptables Maß begrenzen.

Mechanisch problemlos ist eine Realisierung, wenn die Teilsysteme im standardisierten Europakartenformat aufgebaut sind. Durch die festgelegten Teilungseinheiten können Platinen mit einem ganzen Vielfachen der Höheneinheiten verwendet werden, um Teilsysteme mit horizontalen Systembus vertikal zu koppeln. *Bild 10* zeigt als Beispiel ein System mit drei Teilsystemen. Die Teilsysteme werden über ein Buskoppler-Modul lose gekoppelt.

Das Buskoppler-Modul verfügt über einen Kommunikations-Bus, die Pufferung zum Systembus der Teilsysteme, eine Zugriffssteuerung und den RAM-Speicher.

Auf dem Modul ist zusätzlich noch ein Überwachungs-Mikrocomputer zur Erhöhung der Fehlersicherheit und/oder Koordinierung der Teilsysteme einsetzbar.

Sollen alle drei Teilsysteme miteinander über „Message-Channels" verbunden werden, sind hierzu sechs Speicherblöcke (z. B. mit je 256 Byte) im Kommunikationsspeicher notwendig.

2.2 Enge Buskopplung

Die enge Kopplung eines lokalen µP-Bus an den Systembus wird immer dann notwendig, wenn Multi-Master- oder redundante Systeme aufzubauen sind. Die Kommunikation zwischen den Multi-Mikrocomputermodulen wird über gemeinsame Speicherbereiche am Systembus vorgenommen.

Die enge Buskopplung setzt voraus, daß der Systembus multiprozessorfähig ist. Dieses Merkmal läßt es zu, mit mehreren Bus-Mastern an einem Systembus zu arbeiten. Das wichtigste Merkmal der engen Buskopplung ist die Tatsache, daß ein Bus-Master, welcher die Kontrolle über den Systembus besitzt, auch Zugriff zu allen passiven Modulen am Bus hat. Das sind z. B. Speicher-, Ein-/Ausgabe- oder Peripheriemodule. Auch

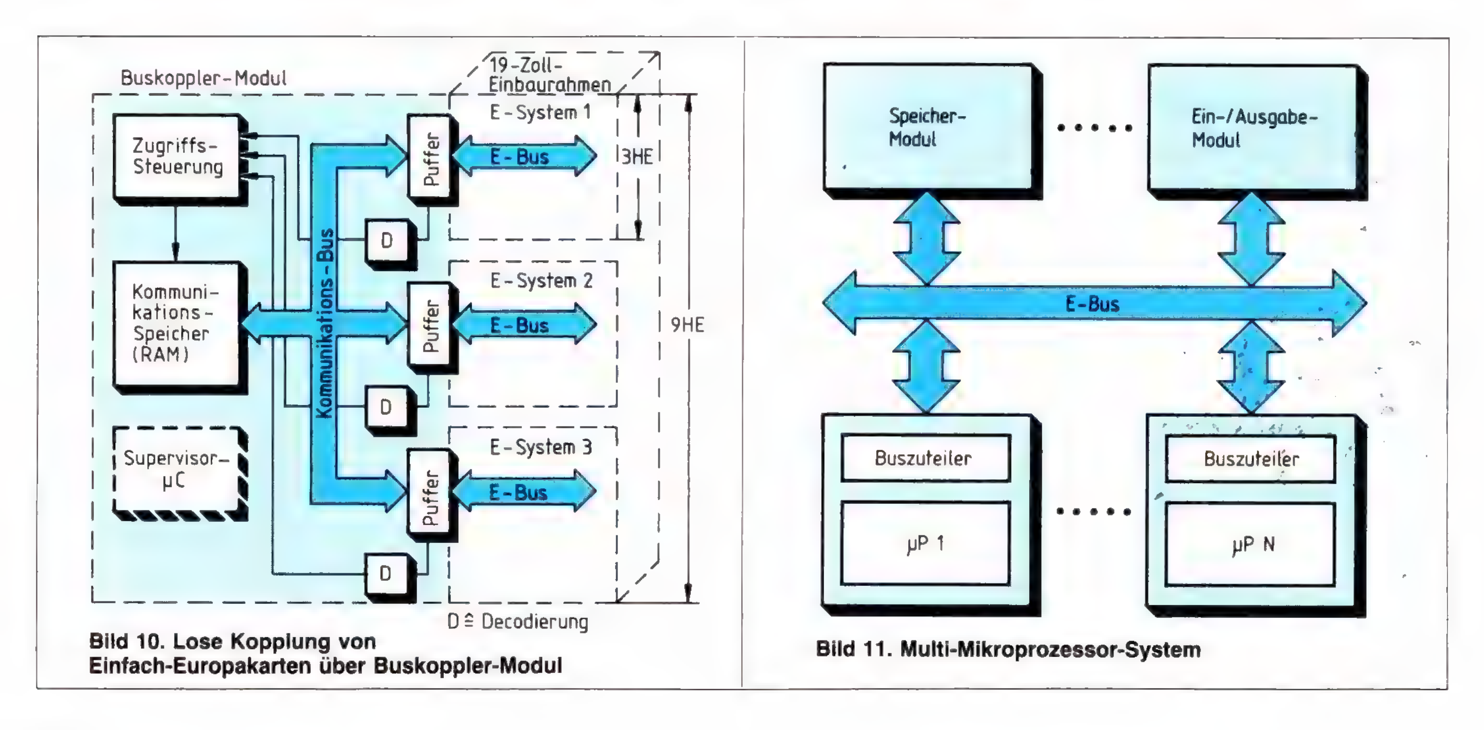


Bild 10. Lose Kopplung von Einfach-Europakarten über Buskoppler-Modul

Bild 11. Multi-Mikroprozessor-System

der Speicher eines Master-Moduls kann, wenn er vom Systembus her adressierbar ist, ein passiver Busteilnehmer sein. Dieses wird z. B. zum Laden größerer Programme von einem Massenspeicher verwendet.

Äußerst bedeutend für die Beurteilung der Möglichkeiten und Grenzen der engen Buskopplung ist, daß der Datendurchsatz des Systembus festliegt. Arbeiten mehrere Master am Bus, so muß die zur Verfügung stehende Buszeit aufgeteilt werden. Die Anzahl der möglichen Master ist daher um so größer, je kleiner die Busbelegung des einzelnen Bus-Masters ausfällt. Die Realisierung einer engen Kopplung von Mikroprozessormodulen (nur Zentraleinheit, ohne lokalen Programm- oder Datenspeicher), wie sie *Bild 11* zeigt, ist bei Verwendung moderner 16-Bit-Mikroprozessoren nicht mehr sinnvoll. Beim Typ TMS 9995 werden z. B. Befehle überlappend verarbeitet, und die Speicherschnittstelle ist zum Teil bereits zu 80 % ausgelastet. Selbst bei einem komplexen Befehl, wie der vorzeichenlosen 16 × 16-Bit-Multiplikation (32-Bit-Ergebnis in 7,67 µs), geht die Belegung höchstens auf ca. 20 % zurück.

Ein Master-Modul am Systembus sollte deshalb über viel lokalen Programm- und Datenspeicher sowie Ein-/Ausgabe verfügen (*Bild 12*). In diesem Fall beschreibt die Bezeichnung Multi-Mikrocomputer-System die Struktur eindeutiger als der häufig verwendete Globalbegriff „Multiprozessorsystem“.

Eine weitere Möglichkeit, die Systembusbelastung zu reduzieren und die Geschwindigkeit des MMC-Moduls zu erhöhen, ist der Einsatz eines schnellen Zwischenspeichers. Dieser sogenannte Cache-Speicher setzt sich zusammen aus einem schnellen RAM-Speicher (Zykluszeit < 100 ns), einem TAG-RAM und einer Cache-Steuerung. Bei Lesezugriffen zum Systemspeicher überprüft die Cache-Steuerung, ob das Speicherwort bereits im schnellen Cache-RAM steht. Ist dieses nicht der Fall, wird das Wort aus dem Systemspeicher gelesen und gleichzeitig im Cache gespeichert. Benötigt der Mikroprozessor erneut das Wort zu einem späteren Zeitpunkt, wird es aus dem schnellen Cache-RAM gelesen. Der Systembus wird nicht erneut belastet. Ein Schreibzyklus bewirkt das Speichern des Wortes im Cache-RAM. Danach schreibt die Steuerung das Wort selbsttätig in den Systemspeicher.

Die Integration von Teilen der Betriebssystemsoftware auf den „On-Chip“-Speicher des Mikroprozessors, wie sie z. B. durch den „Makrostore“ der TMS-99000-Familie [5] möglich wird, bewirkt bei gleichzeitiger Reduzierung der Systembusbelastung ebenfalls eine Erhöhung des Datendurchsatzes. Die enge Systembuskopplung bietet gegenüber der losen Kopplung folgende Vorteile:

- Einfache Erweiterung des Speichers oder der Ein-/Ausgabe durch Standardmodule.
- Zugriff zu allen passiven Modulen.
- Einfache Realisierung redundanter Systeme oder Überwachungsfunktionen.
- Freie Festlegung der Systemstruktur wie Multi-Master oder Master-Slave.

2.2.1 Bussteuerung

Ein Systembus wird durch die Bussteuerung multiprozessorfähig. Die konfliktfreie Steuerung aller möglichen Bus-Master erfordert das Erkennen des Busstatus und Regeln für die Freigabe und die Übernahme der Buskontrolle durch einen Bus-Master. Liegen mehrere Zugriffsanordnungen vor, wird die Buszuteilung über festgelegte Prioritäten geregelt. Beim E-BUS wird die Bussteuerung synchron mit dem Bustakt vorgenommen.

Mit der positiven Bustaktflanke (BUSCLK = Bus-Clock) erfolgt die Freigabe und Übernahme der Buskontrolle durch einen Bus-Master. Der Belegtzustand des E-BUS ist durch eine bidirektionale Busleitung, genannt „BUS BUSY“ (BUSY-), gekennzeichnet.

Ein Bus-Master meldet eine vorliegende Busanforderung durch Deaktivierung (Low-Pegel), der GRANT-OUT-Leitung. Das Signal „ACCESS GRANT IN“ (GRANTIN) des E-BUS dient zur Zuteilung des Buszugriffes (*Bild 15*). Für einen Bus-Master gilt folgende Regel für die Buszuteilung.

– Eine vorliegende Busanforderung wird synchron mit BUSCLK- durch Setzen (Low-Pegel) der GRANTOUT-

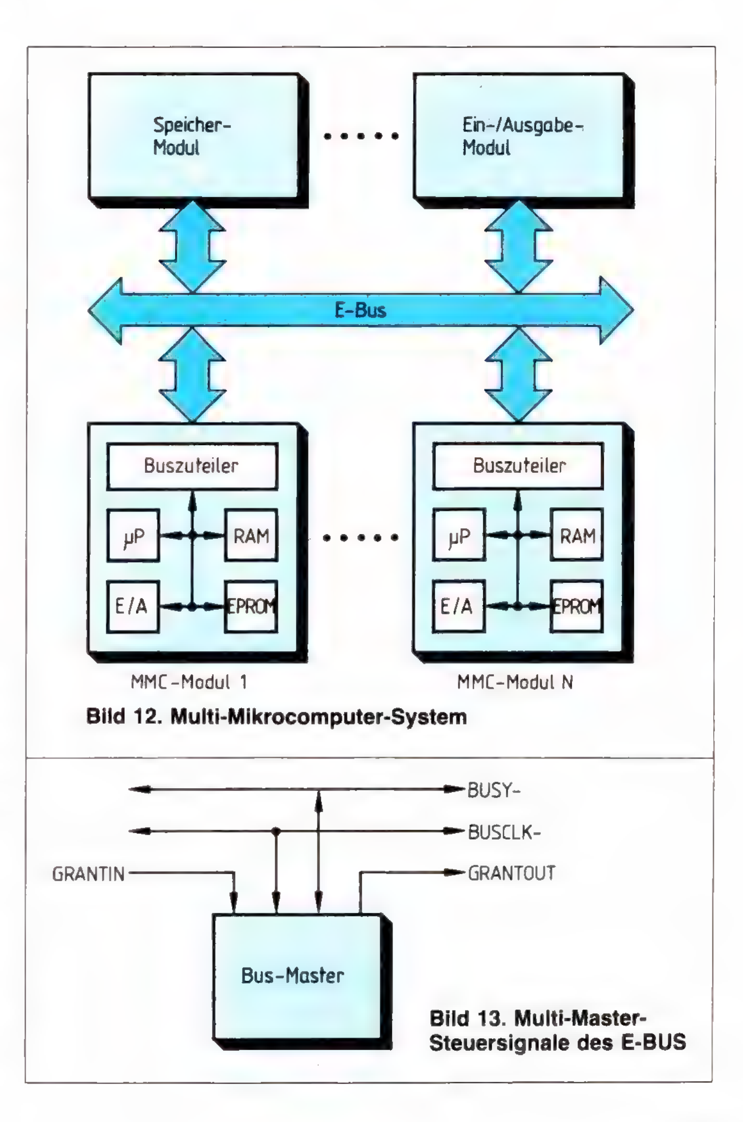


Bild 12. Multi-Mikrocomputer-System

Bild 13. Multi-Master-Steuersignale des E-BUS

Leitung an Bus-Master mit geringerer Priorität oder eine parallele Bussteuereinheit gemeldet. Mit jeder positiven BUCLK-Flanke wird eine Buszuteilung über die Signale „GRANTIN“ und „BUSY-“ durchgeführt.

Die Busanforderung des Masters mit der höchsten Priorität wird durch einen High-Pegel auf der GRANTIN-Leitung quitiert. Ist der E-BUS frei (BUSY- auf High-Pegel), belegt der neue Bus-Master den Bus durch Aktivierung der BUSY- und Deaktivierung der GRANTOUT-Leitung.

Bei einem Bustakt von max. 10 MHz beträgt die Buszuteilungszeit nur 100 ns. Der gegenwärtige Bus-Master unterliegt folgenden Regeln für die Freigabe des E-BUS:

- Liegt keine interne Busanforderung mehr vor oder wird die Buszuteilung entzogen (Low-Pegel auf GRANTIN), so muß der gegenwärtige Bus-Master den E-BUS am Ende der laufenden Busoperation freigeben. Dies erfolgt durch Freigabe der BUSY-Leitung. Ist weiterhin eine interne Busanforderung vorhanden, wird GRANTOUT synchron mit BUSCLK wieder auf Low-Pegel gesetzt und auf eine erneute Buszuteilung gewartet. Die Bussteuereinheit (Arbiter) des Bus-Master darf nur dann von der Freigaberegel abweichen, wenn vom Master ein BUSLOCK-Signal vorliegt. Dieses ist notwendig, um Systemflags (Semaphores) zu testen und zu setzen oder schnelle Blocktransfers durchzuführen.
- Liegen zwei oder mehrere Busanforderungen für den E-BUS vor, so ist nach einem festgelegten Schema der Bus-Master mit der höchsten Priorität auszuwählen. Beim E-BUS ist sowohl die einfache serielle, wie auch die parallele Prioritätensteuerung vorgesehen.

2.2.1.1 *Serielle Prioritätensteuerung*

Bei der seriellen Prioritätensteuerung wird den Steckplätzen eines Einbaurahmens eine feste Priorität zugewiesen. Die Verdrahtung der GRANT-Leitungen erfolgt dabei so, daß GRANTOUT des ersten Bus-Masters mit dem GRANTIN des nächsten verbunden ist. Dadurch erhält der Bus-Master, dessen GRANTIN-Eingang andauernd auf High-Pegel liegt, die höchste Priorität innerhalb der seriellen Kette. Dieses unter dem Namen „Daisy-Chain“ bekannte Zuteilungsverfahren mit statischer Priorität zeichnet sich aus durch:

- geringen Leitungsaufwand,
- dezentrale Buszuteilung mit geringem Hardwareaufwand,
- schnelle Prioritätenfestlegung innerhalb eines BUSCLK-Zyklus,
- geringe Synchronisationsverluste.

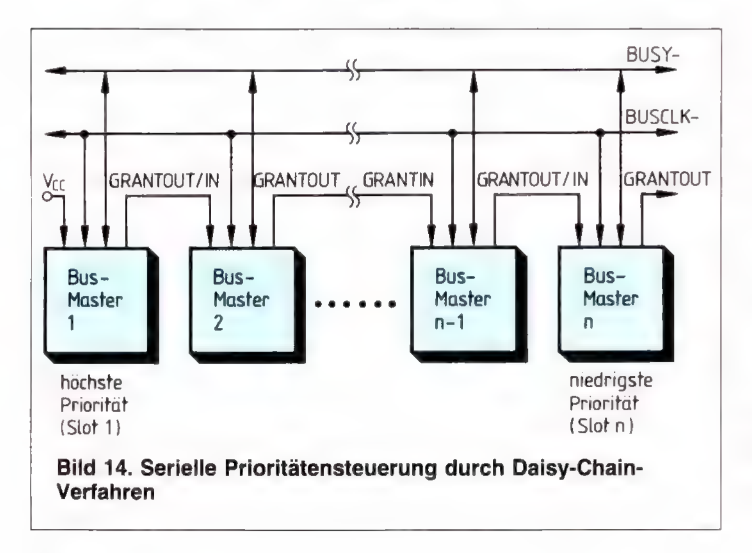


Bild 14. Serielle Prioritätensteuerung durch Daisy-Chain-Verfahren

Als Nachteil des Daisy-Chains wird manchmal die statische Priorität des einzelnen Steckplatzes empfunden. Diese statische Priorität hat zur Folge, daß im Falle einer Überlastung des Systembusses (die Summe der benötigten Buszeit übersteigt die Kapazität des Systembusses) dem Bus-Master mit der geringsten Priorität weniger Buszeit zugeteilt wird. Das heiß, der Datendurchsatz des Bus-Masters mit geringster Priorität wird so weit reduziert, bis der Systembus seine volle Kapazität erreicht hat. Für industrielle Anwendungen ist dies auch durchaus notwendig, da bestimmte Aufgaben innerhalb einer maximalen Zeit auf jeden Fall abgearbeitet werden müssen. Grundsätzlich muß ein MMC-System natürlich so konzipiert werden, daß die zur Verfügung stehende Kapazität des Systembusses ausreicht.

2.2.1.2 *Parallele Prioritätensteuerung*

Der E-BUS sieht für Anwendungsfälle, in denen eine dynamische Prioritätenzuweisung erfolgen soll, die parallele Buszuteilung vor. Die GRANTIN- und GRANTOUT-Leitung jedes einzelnen Bus-Masters wird dabei parallel auf einen speziellen Steckplatz geführt. Dieser spezielle Steckplatz dient zur Aufnahme eines sogenannten „Supervisor-Moduls“ (*Bild 15*).

Am Supervisor-Modul liegen somit die Busanforderungen aller Master-Module parallel an, und die Prioritätenfestlegung kann frei nach den gegebenen Anforderungen gewählt werden. Es läßt sich sowohl die einfache statische, die rotierende und die dynamische Prioritätenzuweisung einsetzen. Die Buszuteilungslogik (Belegung und Freigabe des E-BUS) der Master-Module ist weiterhin dezentralisiert. Lediglich die Prioritätensteuerung wird auf dem Supervisor-Modul zentralisiert. Diese Zentralisierung ist bei der dynamischen Prioritätensteuerung unumgänglich, da sonst keine konfliktfreie Zuteilung gesichert ist.

● *Statische Prioritäten*

Die Buszuteilung nach statischen Prioritäten ergibt, wie beim Daisy-Chain, für jeden Master-Modul eine feste Priorität der Busforderung. Eine direkte Zuordnung der Priorität zur Reihenfolge der Steckplätze ist nicht erforderlich. Bei einer großen Anzahl von Master-Modulen muß anstelle des Daisy-Chains die parallele Priorität-

tensteuerung eingesetzt werden, da die Durchlaufverzögerungszeit der seriellen Kette zu groß wird, um innerhalb eines BUSCLK-Zyklus eine Buszuteilung sicher auszuführen. Die erforderliche Logik zur Prioritätenfestlegung ist einfach mit Standard-LPS-Bausteinen (z. B. Prioritätcodierer SN74LS348) zu realisieren.

● *Rotierende Prioritäten*

In Systemen mit mehreren gleichberechtigten Master-Modulen (z. B. Multi-User-DV-Anlagen) ist es wünschenswert, jedem Master die gleiche Buszeit zur Verfügung zu stellen. D. h., daß der Datendurchsatz aller Master-Module im Falle einer Systemüberbelastung annähernd gleichmäßig reduziert wird. Dieses wird oft als „faire Buszuteilung" bezeichnet, da alle Master-Module einen „fairen" Anteil der Systembuszeit erhalten [6]. Durch die sequentielle Änderung der Prioritätsfolge erhält dann jeder mögliche Bus-Master einmal innerhalb eines Zyklus die höchste Priorität. Bei vier Bus-Mastern z. B. erhält erst Master 1 die höchste Priorität, dann Master 2, 3, 4 und dann wieder Master 1. Die höchste Priorität wird also in einer Kette von einem zum nächst möglichen Bus-Master weitergereicht. Die Realisierung der rotierenden Prioritäten kann vollständig dezentralisiert werden [7, 8]. Nachteil bei der Dezentralisierung ist, daß entweder die Anzahl der Busleitungen [8] oder die Buszuteilungszeit [7] ansteigt.

● *Dynamische Prioritäten*

Soll die Buszuteilung sich ständig den ändernden Systembedingungen anpassen, so kann zu einer dynamischen Prioritätenzuteilung übergewechselt werden. Ziel der dynamischen Zuteilung kann es z. B. sein, die Busanforderungen in der Reihenfolge ihres Auftretens zu bestätigen (FIFO-Prinzip). Alternativ kann auch eine programmierbare Prioritätsfolge oder eine Kombination aus festen Prioritäten und FIFO-Prinzip gewählt werden. Treten mehrere Busanforderungen gleichzeitig auf, erfolgt die Zuteilung wieder an Hand einer festen oder programmierbaren Sequenz. Die Unterschiede der drei möglichen Prioritätensteuerungen in der Buszuteilung zeigt *Bild 16* am Beispiel eines Systems mit vier MMC-Modulen. Bei festen Prioritäten der Master (Priorität 1, 2, 3 und 4) unterbricht jede Busanforderung mit höherer Priorität einen Master mit geringerer Priorität. Für den MMC-Modul 1 bedeutet dies, daß es den E-BUS immer mit minimaler Verzögerungszeit zugeteilt bekommt. Die Zuteilung nach rotierenden Prioritäten, mit der Sequenz 1, 2, 3 und 4 hat zur Folge, daß jede Busanforderung so lange verzögert wird, bis der Master die höchste Priorität erhält. Die maximale Antwortzeit setzt sich aus der Summe der Buszeiten der anderen möglichen Bus-Master zusammen.

Das FIFO-Prinzip der dynamischen Zuteilung wurde in Bild 18 vorausgesetzt. D. h., alle Busanforderungen werden in der Reihenfolge ihres Erscheinens beantwortet (1-2-4-3 usw.). Setzt man die Buszeiten der vier MMC-Module zu 100 % und vergleicht die drei Arten der Buszuteilung in Bild 16, so fällt auf, daß bei festen Prioritäten für den Master mit der höchsten Priorität definierte Zuteilungszeiten erreicht werden kann.

Erkauft wird dieser Vorteil durch Verzögerung der Master mit geringerer Priorität. Ist die „Überlastung" des Systembus nur kurzzeitig, sind die Unterschiede in der Buszuteilung nur in der Reihenfolge zu sehen. Die Buszuteilung nach statischen Prioritäten kann auch in einer Form erfolgen, welche einen blockorientierten Buszugriff für jeden Bus-Master ermöglicht. Ein Master höherer Priorität übernimmt die Buskontrolle erst, wenn der Buszugriff des gegenwärtigen Bus-Masters abgeschlossen ist. Die wesentlichen Vorteile der parallelen Busprioritätenzuteilung durch ein Supervisor-Modul sind die Möglichkeit der Überwachung und Steuerung der Busaktivitäten. Bus-Master können durch WATCH-DOG-Schaltungen [9, 10] überwacht und im Fehlerfall vom E-Bus getrennt werden. Weitere Vorteile der parallelen Prioritätensteuerung sind:

- Schnelle Buszuteilung innerhalb eines BUSCLK-Zyklus;
- Freie Wahl der Prioritätenzuweisung (feste, rotierende oder dynamische Prioritätenzuweisung);
- Einfache Testmöglichkeiten während der Entwicklungs- und Betriebsphase des MMC-Systems.

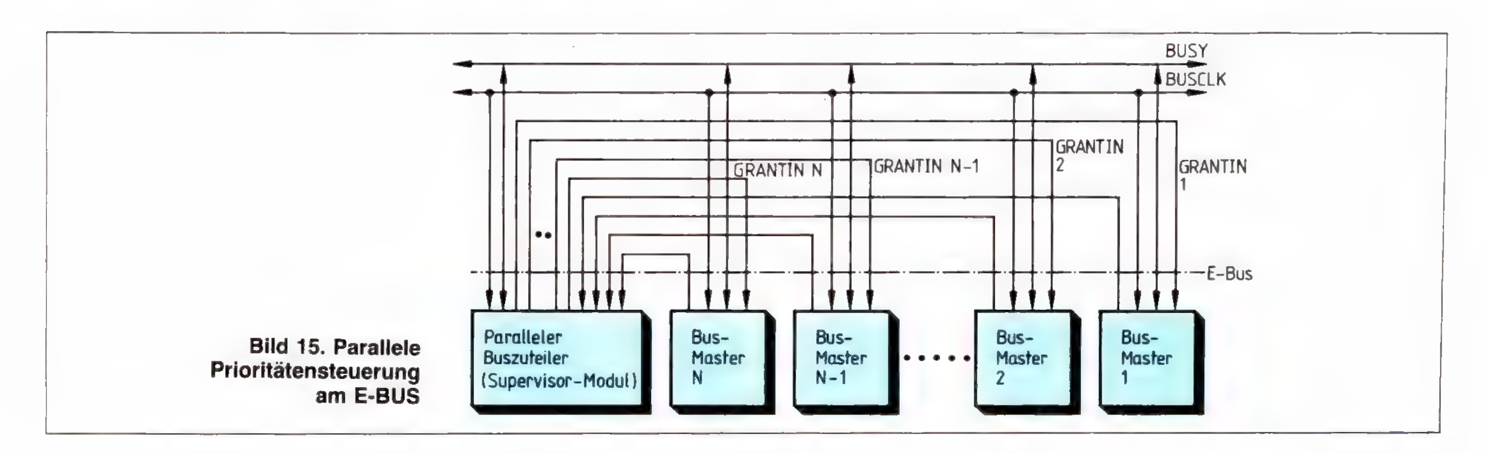


Bild 15. Parallele Prioritätensteuerung am E-BUS

Für die genannten Vorteile muß der Systementwickler den etwas höheren Hardware-Aufwand des Supervisor-Moduls in Kauf nehmen. Beim E-BUS sind für die parallele Prioritätensteuerung bis zu acht Bus-Master, vier DMA- und zwei Interrupt-Kanäle innerhalb eines 19-Zoll-Steckrahmens vorgesehen.

3 Praktischer Einsatz der Systembuskopplung

Die in einem Mikrocomputersystem verfügbare Rechenleistung kann oft schon durch den Einsatz eines weiteren MMC-Moduls entscheidend verbessert werden. Es ist dabei nicht unbedingt notwendig, komplexe Multi-Mikrocomputerstrukturen aufzubauen, wenn die Teilung der Systemfunktion auf ein Doppel-Prozessorsystem ausreicht. Dieses ist der Fall wenn:

- Zeitprobleme nur bei einer bestimmten Funktion vorliegen, oder
- einfache Ausfallüberwachung gewünscht wird, oder
- die Teilung der Systemfunktion für die Softwareerstellung günstig wird.

Als Beispiel soll hier kurz ein Doppel-Prozessorsystem beschrieben werden, auf welches alle genannten Punkte zutreffen. Die Baugruppe des Systems besitzt ein µC-Modul, das vorwiegend in BASIC programmiert wird. Außerdem beinhaltet es ein an den E-BUS eng gekoppeltes MMC-Modul. Die zeitkritischen Operationen (z. B.) dreidimensionale Bahnberechnungen) werden in Assembler programmiert und das Programm ist vorwiegend im lokalen EPROM-Bereich des MMC-Moduls gespeichert. Der Assembler-Rechner bedient alle zeitkritschen Ein-/Ausgabeoperationen und erhält seine Steuerparameter vom BASIC-Rechner. Der Datenaustausch läuft über das gemeinsame Speichermodul am E-BUS, welches auch das BASIC-Anwenderprogramm aufnimmt. Die Belastung des Systembus durch den BASIC-Rechner ist nach vorliegenden Erfahrungen kleiner als 10 %, da sich der BASIC-Interpreter im lokalen Speicher befindet. Dadurch steht dem Assembler-Rechner mehr als genügend Buszeit zur Verfügung für die Ausführung umfangreicher Ein-/Ausgabeoperationen.

Eine zusätzliche Steigerung der Systemleistung durch Hinzufügen weiterer BASIC- oder Assembler-Rechner ist möglich.

Literatur

[1] *Bellm, H.* und *Thinschmidt, H.*: Ein Multi-Mikrocomputersystem am Arbeitsplatz, Teil 2: Bus-Strukturen für Mehrrechnersysteme. ELEKTRONIK 1979, H. 20, S. 73...77.

[2] *Conrad, M., Vincent, G.*: Funktionsorientierte Architektur mit Software-Komponenten. ELEKTRONIK-Sonderheft Softwarewerkzeuge. Franzis-Verlag, München.

[3] *Althoff, H. G.*: Multiprozessorbus-System für Europakarten. ELEKTRONIK 1980, H. 18, S. 57...62.

[4] „E-Bus Systemdesign Handbuch", MP407, Texas Instruments Deutschland GmbH.

[5] *Laffitte D., Guttag, K.*: Fast on-chip memory extends 16-Bit family's reach. Electronics Feb. 24, 1981, S. 157...161.

[6] *Soe Jojberg, K.*: Queue Handling Arbiter Solves Shared Resources Conflicts. Computer Design, Nov. 1979, S. 129...135.

[7] *Angelé, G.*: Algorithmus und Implementierung eines dezentralen fairen und fehlertoleranten Busarbiters. ELEKTRONIK 1981, H. 9, S. 79...84.

[8] *Färber, G.*: Ein dezentralisierter fairer Bus-Arbiter. ELEKTRONIK 80, H. 8, S. 65...68.

[9] *Will, B.*: Erhöhte Ausfallsicherheit bei Systemen mit verteilter Intelligenz. ELEKTRONIK 1979, H. 7, S. 69...73.

[10] *Huse, H.*: Watch-Dog-Schaltungen erkennen µP-Systemstörungen. ELEKTRONIK 1980, H. 4, S. 92...94.



Ing. (grad.) Horst Huse, Jahrgang 1945, studierte in Rüsselsheim allgemeine Elektrotechnik. Seit 1973 ist er Mitarbeiter der Firma Texas Instruments in Freising und derzeit Leiter des Applikationslabors für Mikroprozessor- und Industrieanwendungen.
Hobbys: Fotografieren und Lesen
Diensttelefon: (0 81 61) 80 40 41
Privattelefon: (0 81 61) 8 46 54

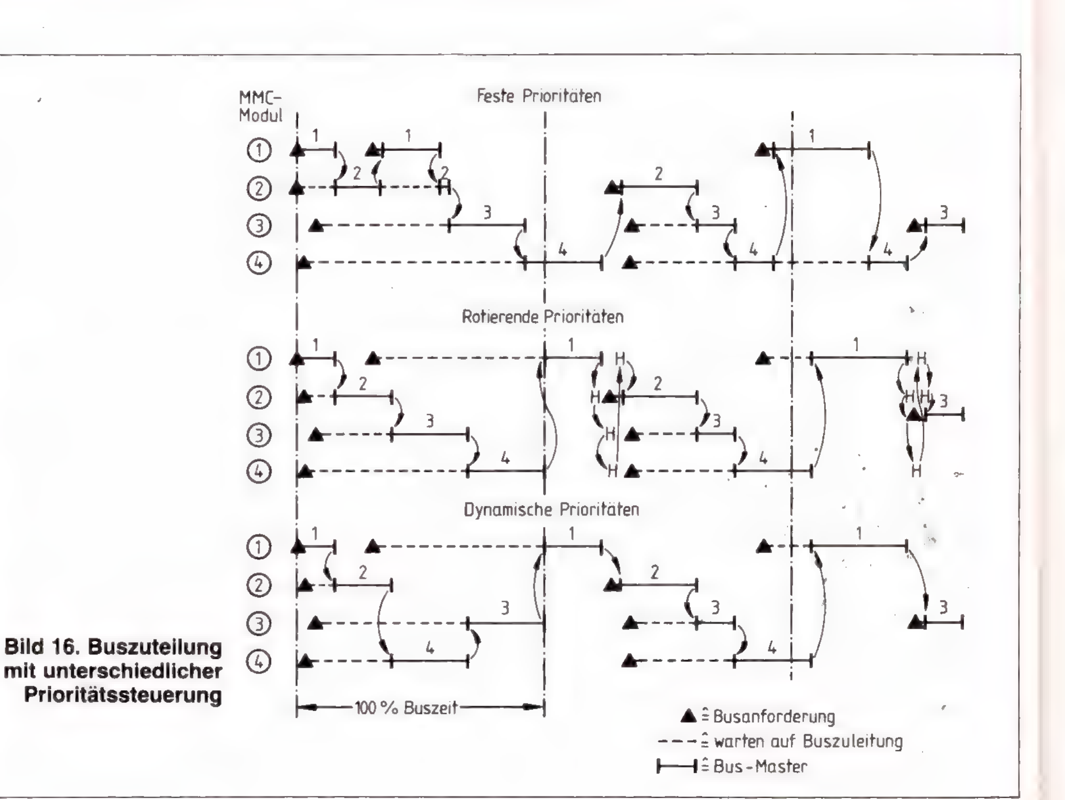


Bild 16. Buszuteilung mit unterschiedlicher Prioritätssteuerung

Ing. (grad.) Karl-Heinz Wendel

Ein modulares, hierarchisch strukturiertes Multi-Mikrocomputer-System

Bei der klassischen Rechnerstruktur läßt sich der Durchsatz nicht beliebig steigern. Zur Lösung dieses Problems bieten sich Multi-Mikrocomputer-Systeme an. Hier strebt man als Idealfall an, daß die Summe der Einzeldurchsätze gleich dem Gesamtdurchsatz ist. Inwieweit man das erreicht, hängt im wesentlichen von der Art und Anzahl der gemeinsamen Betriebsmittel (Bus, Speicher) ab. Dieser Gesichtspunkt wurde beim nachfolgend beschriebenen System besonders berücksichtigt.

1 Systemkonzept

Die Rechnerkonzeption entspricht einem modularen, (baumförmig) strukturierten Multi-Mikrocomputer-System (*Bild 1*). Das System ist in die Gruppe der Computer einzuordnen, die man unter der Bezeichnung „Multiple Instruction Stream/Multiple Data Stream" zusammenfaßt. Als CPU eignen sich die Mikroprozessoren TMS9900, TMS9980/81 und TMS9940 [8, 9, 10]. Jeder Mikrocomputer verfügt über private Speicher und hat keinen direkten Zugriff zum Speicher eines anderen (*Bild 2*, [7]). Es sind keine konkurrierenden Zugriffe auf gemeinsame Betriebsmittel (Bus, Speicher) erforderlich, Verklemmungssituationen (*Deadlocks*) können somit vermieden werden. Dieses Konzept ist besonders bei steigender MC-Zahl vorteilhaft, da sich im allgemeinen ein allen Mikrocomputern gemeinsames Betriebsmittel als Flaschenhals erweist.

Bei der Systemauslegung gilt es, den zu bearbeitenden Gesamtprozeß mit einer Top-Down-Strategie in sinnvolle Teilprozesse zu zerlegen und diese in verschiedenen Betriebsmitteln weitgehend autonom zu realisieren. Die Aufgabenteilung führt zu einer Parallelisierung der Teilprozesse und zu einem praktisch beliebig steigerbaren Durchsatz des Gesamtsystems. Daneben sollte bei der Zerlegung in Teilprozesse auf einen möglichst geringen Datenverkehr zwischen den einzelnen Einheiten geachtet werden. Datenreduktion an geeigneter Stelle und, soweit machbar, lediglich die Integration von Teilergebnissen in der übergeordneten Hierarchie-Ebene sind hierzu Lösungsansätze. Des weiteren können bereits untergeordnete µC-Stationen über Schnittstellen zur „Umwelt" verfügen, um den Datenfluß innerhalb des Gesamtsystems zu minimieren. Die feste Zuordnung der Teilprozesse zu den einzelnen Mikrocomputern führt zu einem relativ einfachen Betriebsystem, geringerem Schnittstellenaufwand und zu größerer Transparenz, als dies bei wahlfreier Zuordnung möglich wäre. In allgemeiner Anwendung ist durch den modularen Aufbau eine optimale Anpassung an die problemspezifischen Erfordernisse gewährleistet.

Die hierarchische Struktur nach dem Master-Slave-Prinzip ergibt sich aus bestimmten Interrupt-Fähigkeiten der untergeordneten µC-Stationen. Die Slave-Station kann von Seiten des Masters zu einer um wenige µs Reaktionszeit verzögerten Kontaktaufnahme veranlaßt

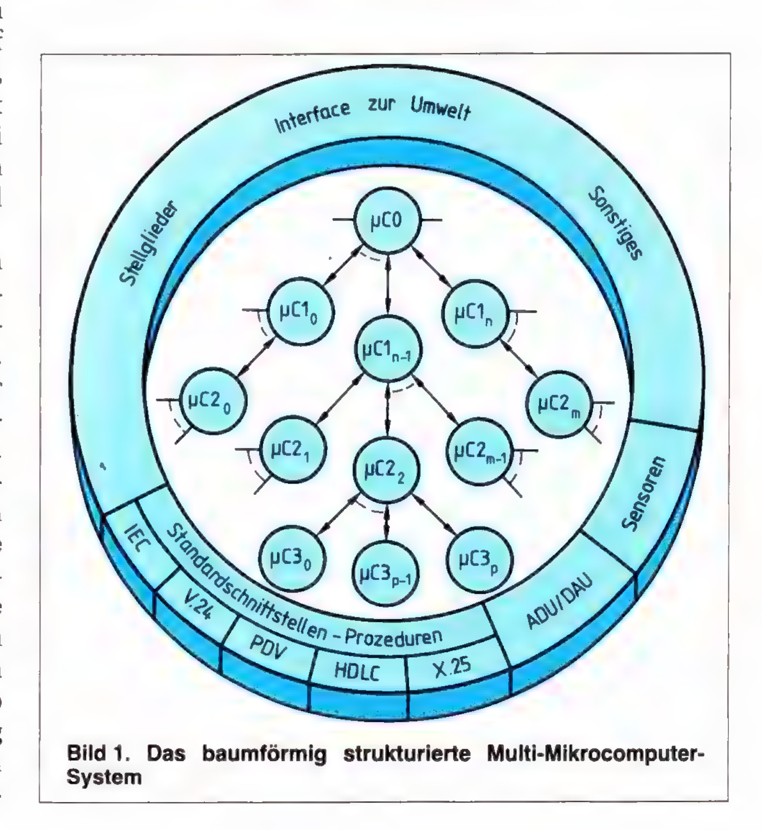


Bild 1. Das baumförmig strukturierte Multi-Mikrocomputer-System

werden (*Interrupt Context Switch* bei TMS9900 ca. 7 µs). Die Slave-Station ihrerseits signalisiert über ein definiertes Request-Bit die Bereitschaft zur Kommunikation. Dieser Anforderung muß nicht zwingend von Seiten des Masters in einer bestimmten Zeit nachgekommen werden. Somit ergibt sich, bezüglich der Reaktionszeit, eine relativ enge Kopplung in der Hierarchie von oben nach unten und eine lose Kopplung von unten nach oben.

2 Kommunikation zwischen den µCs

Über eine rechnerspezifische CRU (*Communication Register Unit*, [12]) wird nach der Handshaking-Methode die Kommunikation eingeleitet, unterhalten und beendet (*Bild 3*). Über diese Schnittstelle werden in Richtung Slave Befehle und Daten und in Richtung Master Statusmeldungen und Daten gesendet.

Befehle und Statusmeldungen sind bitweise codiert. Sie werden mit SBO-/SBZ-Befehlen (*Set Bit to One, Set Bit to Zero*) in einem µC generiert und per TB-Befehl (*Test Bit*) im anderen abgefragt.

Die Daten-Ports werden mit den Mehrbitbefehlen LDCR (*Load Communication Register*) und STCR (*Store Communication Register*) geladen bzw. gelesen. Wegen des bitseriellen Übertragungsmodus an der CRU-Schnittstelle werden für die µC/µC-Schnittstelle lediglich die CRU-Steuersignale CRUCLK, CRUOUT, CRUIN sowie eine, von der Breite der Daten-Ports und des Befehls-/Status-Vorrats abhängige Anzahl an Adreßleitungen benötigt. Es ist also nicht erforderlich, den Datenbus oder den kompletten Adreßbus der Mikrocomputer zur gemeinsamen Schnittstelle herauszuführen. Dies hat neben der geringeren Zahl benötigter Leitungen und zugehöriger Signalpuffer eine Erhöhung der Störsicherheit zur Folge.

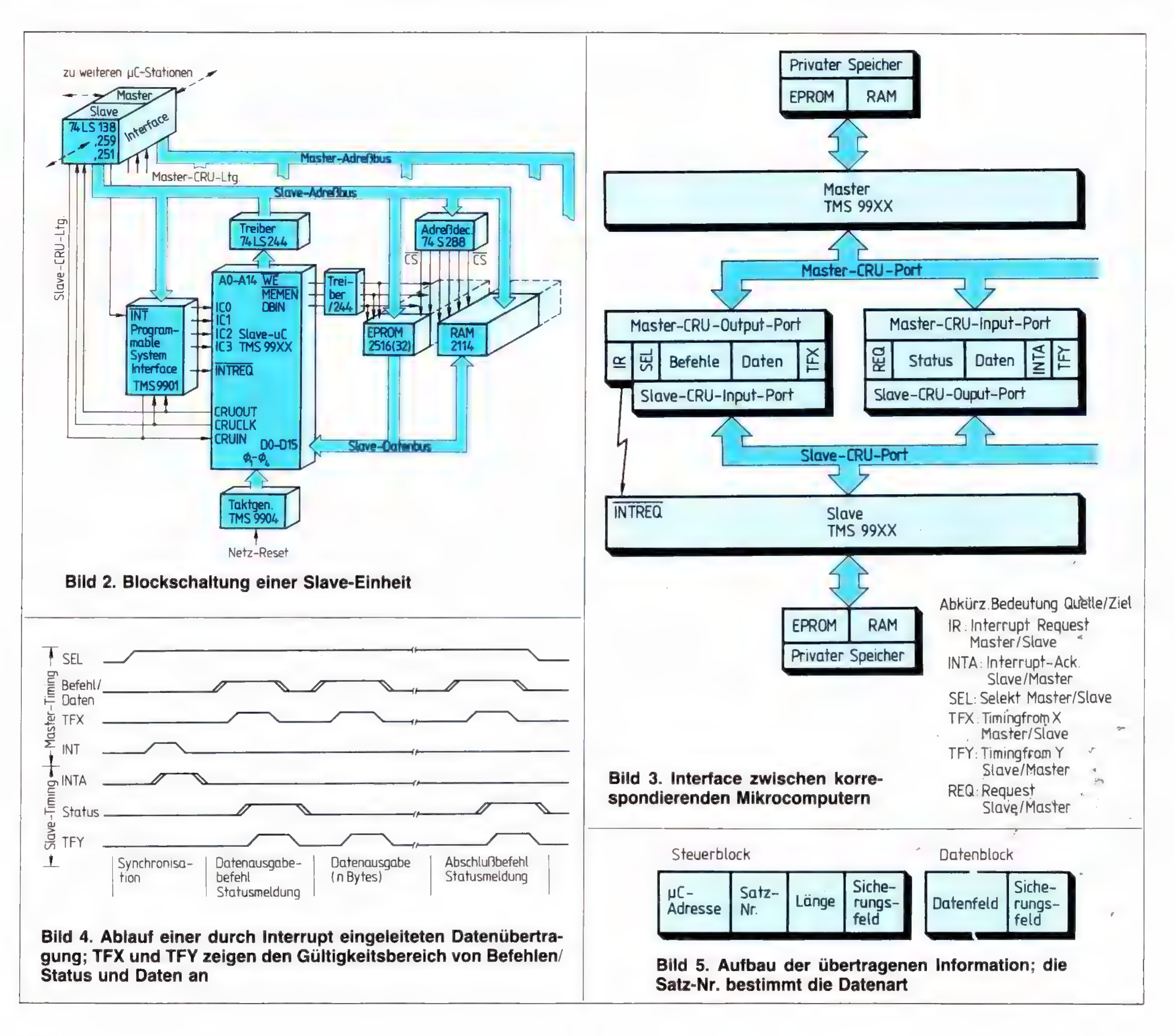


Bild 2. Blockschaltung einer Slave-Einheit

Bild 3. Interface zwischen korrespondierenden Mikrocomputern

Bild 4. Ablauf einer durch Interrupt eingeleiteten Datenübertragung; TFX und TFY zeigen den Gültigkeitsbereich von Befehlen/Status und Daten an

Bild 5. Aufbau der übertragenen Information; die Satz-Nr. bestimmt die Datenart

Die Datenübertragungsrate einschließlich der die Übertragung vorbereitenden und abschließenden Befehlsausgaben und Statusmeldungen liegt bei 70 kBd. Falls erforderlich, kann mit einer Verbreiterung der Daten-Ports auf 16 Bit, unter Verwendung der vorliegenden Prozeduren, die Übertragungsrate auf etwa 110 kBd gesteigert werden.

Wird von Seiten des Masters eine Kommunikation gewünscht, erfolgt der Anstoß hierzu per Interrupt: Der Master setzt ein Interface-Bit (IR), das im Slave auf einen Interrupt-Eingang des TMS9901 [11] geführt ist. Die erfolgte Reaktion zeigt der Slave mit einem Interrupt-Acknowledge-Bit an. Der Slave seinerseits meldet sich über ein Request-Bit (REQ). Dieses Bit wird im Polling-Mode vom Master abgefragt und, abhängig vom aktuellen Betriebszustand, erfolgt eine entsprechende Reaktion des Masters.

3 Software

Der Software-Anteil ist, vornehmlich mit zunehmender Komplexität des auszuführenden Gesamtprozesses, im Entwicklungsaufwand deutlich höher anzusetzen als die Hardware-Komponente. Das Bindeglied zwischen Hardware und Software stellen die Übertragungsprozeduren dar (*Bild 4*). „Time-out"-Kriterien überwachen den Ablauf der Prozeduren und können im Fehlerfall zu einer Wiederholung bzw. bei Häufung der Übertragungsfehler zu Fehlermeldungen und zum Aufruf eines Selbsttestprogramms führen.

Auf diese Weise kann durch Ausnutzung der verteilten Intelligenz, ohne zusätzlichen Hardware-Aufwand (Speicherbedarf für das Testprogramm vernachlässigt) und unter der Voraussetzung, daß die höchste µC-Station fehlerfrei ist, eine defekte µC-Station bzw. Übertragungsstrecke erkannt werden. Diese Voraussetzung entfällt, wenn auf der obersten µC-Station ein Ersatz-Mikrocomputer als „kalte Reserve" installiert ist. Dieser tritt, in Verbindung mit einer m-aus-n-Entscheidungslogik der untergeordneten Mikrocomputer, im Fehlerfall an die Stelle der defekten Einheit. Der Aufwand steigt jedoch erheblich und ist in der Regel nur in Systemen mit hoher Gesamtverfügbarkeitsforderung (Reaktortechnik, Prozeßautomatisierung usw.) zu vertreten.

Der Informationsfluß zwischen korrespondierenden Mikrocomputern erfolgt von Speicher zu Speicher, byteseriell als Blocktransfer mit einem in der Länge variablen Datenfeld. Nach der Synchronisierung wird von Seiten der übergeordneten Station zunächst ein Steuerblock gesendet. Er beinhaltet die Adresse des selektierten Slaves (redundant zum Select-Bit), eine Record-Nr., welche die Art des codetransparenten Datenfeldes bestimmt, einen Parameter für die Länge des Datenfeldes und ein Sicherungsfeld. Das Sicherungsfeld wird nach einem CRC-Verfahren gebildet und sichert den Steuerblock ab (*Bild 5*). Mit einer Statusmeldung zeigt der Slave, ob er die vom Master im Steuerblock angegebenen Informationen senden oder empfangen kann.

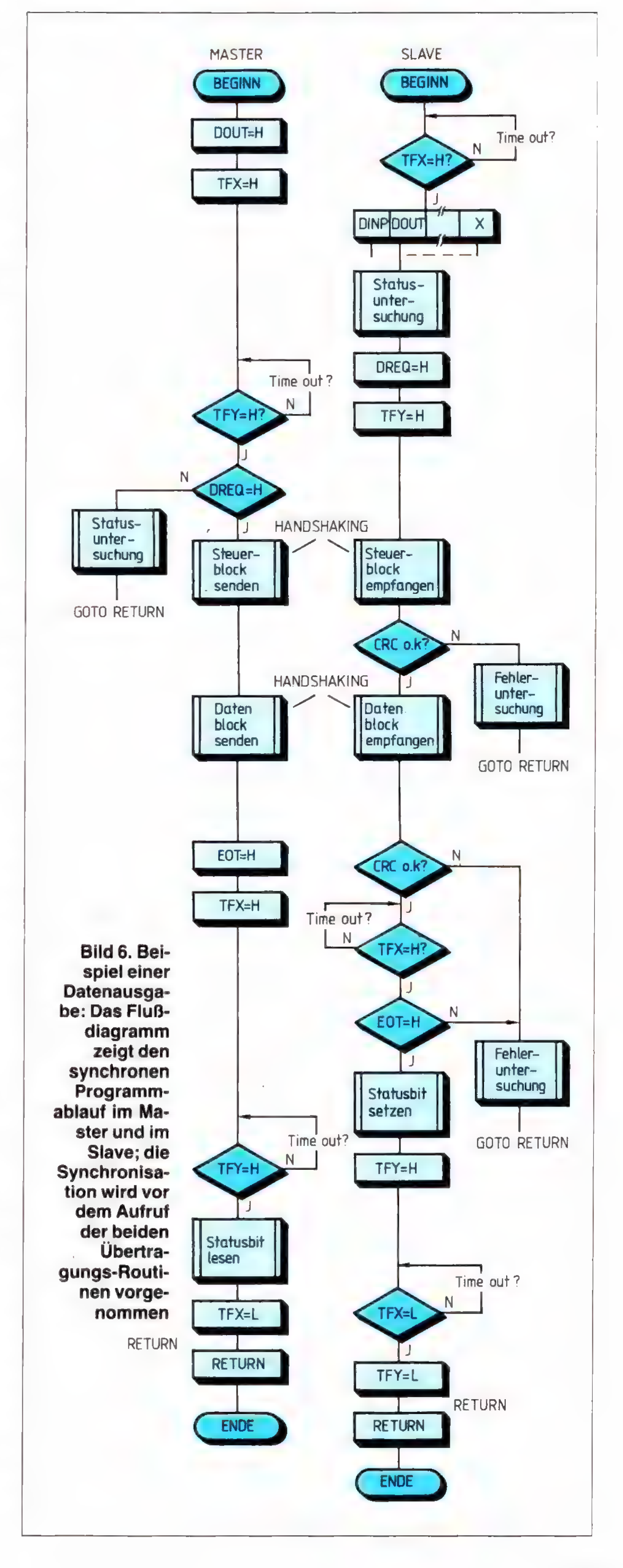


Bild 6. Beispiel einer Datenausgabe: Das Flußdiagramm zeigt den synchronen Programmablauf im Master und im Slave; die Synchronisation wird vor dem Aufruf der beiden Übertragungs-Routinen vorgenommen

Nach entsprechender Statusmeldung erfolgt der eigentliche Datentransport. Die Übertragungsrichtung ist hierbei generell aus der Sicht der jeweils übergeordneten Station festgelegt. Abgeschlossen wird die Datenübertragung mit einem Abschlußbefehl (EOT) des Masters und einer Statusmeldung des Slaves. *Bild 6* zeigt als Beispiel eine Datenausgabe.

3.1 Befehle

Befehle werden von der Master-Station gesendet und von der Slave-Station mit einer Statusmeldung beantwortet. Durch die Art der in der Slave-Station erzielten Reaktionen, kann man unterscheiden zwischen Befehlen, die einen Informationsfluß zwischen Master und Slave bewirken und solchen, die den Slave veranlassen, seinerseits ihm untergeordnete Mikrocomputer und/oder Schnittstellen anzusprechen. Zur letztgenannten Gruppe zählen auch die vom Anwender problemspezifisch zu definierenden Befehle, die zur eventuellen Erweiterung des Befehlsvorrats (*Tabelle 1*) zur Verfügung stehen.

Statusmeldungen geben der übergeordneten Station Aufschluß über den aktuellen Betriebszustand der selektierten, untergeordneten Station. Sie können neben dem „normalen" Zustand auch bestimmte Hastsituationen (z. B. Puffer voll) oder Fehlerzustände aufzeigen. Freie CRU-Ausgabe-Ports bieten dem Anwender darüber hinaus die Möglichkeit, den vorhandenen Statusvorrat problemspezifisch zu erweitern. Diese Statusinformationen (*Tabelle 2*) sind nicht zu verwechseln mit den prozessoreigenen Statusbits.

Tabelle 1. Befehlsvorrat

Bezeichnung	Wirkung
NOP No Operation	Slave schickt Statusmeldung
DINP Data Input	Master leitet eine Dateneingabe ein
DOUT Data Output	Master leitet eine Datenausgabe ein
EOT End of Transfer	Master schließt eine fehlerfreie Datenübertragung ab
TERR Transfer Error	Master beendet eine fehlerhafte Datenübertragung
CLRST Clear Status	Veranlaßt den Slave, seine Statusbits in einen def. Grundzustand zu versetzen
RES Reset	Versetzt den Slave in einen Anfangszustand

Tabelle 2. Statusinformationen

Bezeichnung	Bedeutung
CREQ Command Request	Slave erwartet einen Befehl
DREQ Data Request	Slave ist zur Datenübertragung bereit (nach DINP oder DOUT)
DFLG Data Flag	Slave fordert Beendigung der Datenübertragung nach einem Fehler an
ATT Attention	Slave weist auf eine Hastsituation hin
JMP Jump	Slave fordert den Master zu einer Programmverzweigung auf

Ing. (grad.) Karl-Heinz Wendel war nach Elektromechaniker-Lehre und Berufsaufbauschule (Abendklassen) von 1964 bis 1968 im Prüfmittelbau bei Philips Electrologica, Siegen, angestellt. Im Anschluß an ein Studium der Elektrotechnik (Nachrichtentechnik) an der Staatl. Ing.-Schule Siegen arbeitete er bei derselben Firma von 1971 bis 1978 im Bereich der mittleren Datentechnik. Seit 1978 ist er auf dem Sektor Medizinelektronik mit Systementwicklungen beschäftigt.
Hobbys: Sport, Schach
Privattelefon: (0 76 41) 5 19 37
Diensttelefon: (07 61) 4 01 13 49



4 Anwendung

Ein Multi-Mikrocomputer-System erscheint dann angezeigt, wenn die Erfordernisse eines Gesamtprozesses die Rechenleistung einer einzelnen CPU übersteigen oder in dessen Grenzbereich liegen. Dies kann besonders in rechenintensiven Gebieten wie numerischer Mathematik, Simulationstechnik Prozeßautomatisierung und Mustererkennung, die mit hohen Echtzeitforderungen verknüpft sind, der Fall sein. Multi-Mikrocomputer-Systeme dringen hier, je nach Ausbaustufe, in den Leistungsbereich von Minicomputern und Großrechenanlagen vor. Zukünftig wird ein Hauptaugenmerk auf die Entwicklung „verteilter Betriebssysteme" zu richten sein. Unter Beachtung von Auf- und Abwärtskompatibilität der Einzelkomponenten sowie der Software-Portabilität bieten Multi-Mikrocomputer-Systeme eine wirtschaftliche Lösung der genannten Problemstellungen.

Literatur

[1] *Faggin, F.*: VLSI verändert Computer-Strukturen. ELEKTRONIK 1978, H. 12, S. 57...61.
[2] *Raphael, A.*: Mehrfachverarbeitung mit Multiprozessorsystemen. ELEKTRONIK 1978, H. 11, S. 84...87.
[3] *Bode, A.*: Bit-Slice-Architekturen: Auswirkung des Mangels an Kommunikationswegen auf die Struktur von Mikroprozessoren. NTG 62, Fachtagungsbericht.
[4] *Feissel, W.*: Der Einfluß eines Pufferspeichers auf die Operationsgeschwindigkeit kommerzieller DV-Anlagen. GI-NTG, Fachtagungsbericht 1974: Struktur und Betrieb von Rechensystemen.
[5] *Lehner, M., Horneber, E.-H.*: Strukturiertes Multimikroprozessor-System SMS 201. Siemens Forsch.- u. Entwicklungsb. 1980/2, Bd. 9, S. 88...93.
[6] *Kudielka, V.*: Parallelverarbeitung und modularer Systemaufbau, GI-NTG, Fachtagungsbericht 1974: Struktur und Betrieb von Rechensystemen.
[7] *Schmidt, B.*: Betriebssystemstruktur für eine Mehrprozessorkonfiguration mit privaten Speichern. NTG 62, Fachtagungsbericht.
[8] TMS 9900 Microprocessor Data Manual. Texas Instruments.
[9] TMS 9981 Microprocessor Data Manual. Texas Instruments.
[10] TMS 9940 Microprocessor Data Manual. Texas Instruments.
[11] TMS 9901 Programmable System Interface. Texas Instruments.
[12] *Höger, W., Tinul, W.*: Meßwertverarbeitung mit einem Mikrorechner. ELEKTRONIK 1980, H. 6, S. 61...64.

Prof. Dipl.-Ing. Günter Schmitt
Dr. rer. nat. Wolf Dieter Weiß

Universelles Bussystem für verschiedene Mikroprozessortypen

In den Anfängen der Mikrocomputertechnik wurde die Software ausschließlich für den Mikroprozessor geschrieben, der in der zu entwickelnden Schaltung vorhanden ist. Mit der Einführung immer leistungsfähigerer Prozessoren entstand das Problem der Übertragbarkeit bestehender Programme auf neue Systeme, das sich auch durch den Einsatz problemorientierter Sprachen und „universeller" Cross-Entwicklungssysteme nicht befriedigend lösen läßt. Bestehen zum Beispiel für ein größeres Entwicklungsprojekt bereits Teilprogramme jedoch verschiedener Prozessoren oder können Softwarepakete anderer Prozessorfamilien fertig erworben werden, so bleibt nur eine Lösung, nämlich verschiedene Prozessoren in einem System mit gemeinsamen Speicher- und Peripheriebereichen laufen zu lassen. Die vorliegende Arbeit beschreibt ein Bussystem, das mit unterschiedlichen Prozessortypen betrieben werden kann und das darüber hinaus sowohl in der Entwicklungs- als auch in der Anwendungsphase verwendbar ist.

1 Aufgabenstellung

An ein universelles Bussystem werden folgende Anforderungen gestellt:
- Betrieb mit unterschiedlichen Prozessoren wie z. B. der 8080- und der 6800-Familie;
- Betrieb mit Prozessoren unterschiedlicher Bitbreite wie z. B. mit den Standard-8-Bit-Prozessoren und den neuen 16-Bit-Prozessoren;
- Multiprozessorfähigkeit mit software- und hardwaregesteuerter Busübergabe;
- auch für 8-Bit-Prozessoren Speichererweiterung über die Adressierungsfähigkeit von 64 KByte hinaus;
- die Speicher- und Peripheriekarten müssen ohne Umschaltung für alle Prozessortypen verwendbar sein;
- das Bussystem muß sowohl für Entwicklungen als auch für Anwendungen einsetzbar sein.

Bei einem solchen universellen Bussystem kann jeder der Prozessoren einzeln oder gemeinsam mit anderen sowohl mit den Speichern als auch mit der Peripherie und den Massenspeichern zusammenarbeiten. Daher ist es möglich, das System in der Entwicklungsphase mit verschiedenen Betriebssystemen zu betreiben und in der Anwendungsphase Programme verschiedener Prozessortypen miteinander zu mischen. Die Entwicklungs-Hilfsmittel wie Massenspeicher und Interface-Schaltungen für Drucker und EPROM-Programmiergerät sind für jeden Prozessor verwendbar. Dies ermöglicht eine einfache Umrüstung des Systems auf neue Prozessortypen. Nach der Entwicklungsphase werden die Entwicklungshilfsmittel (Monitorkarte, Massenspeicher) entfernt und das System dient dann als Anwendungs-Computer.

2 Zeitliche Steuerung

Unter Bus-Timing wird hier die zeitliche Steuerung der Datenübergabe zwischen Prozessor und Speicher

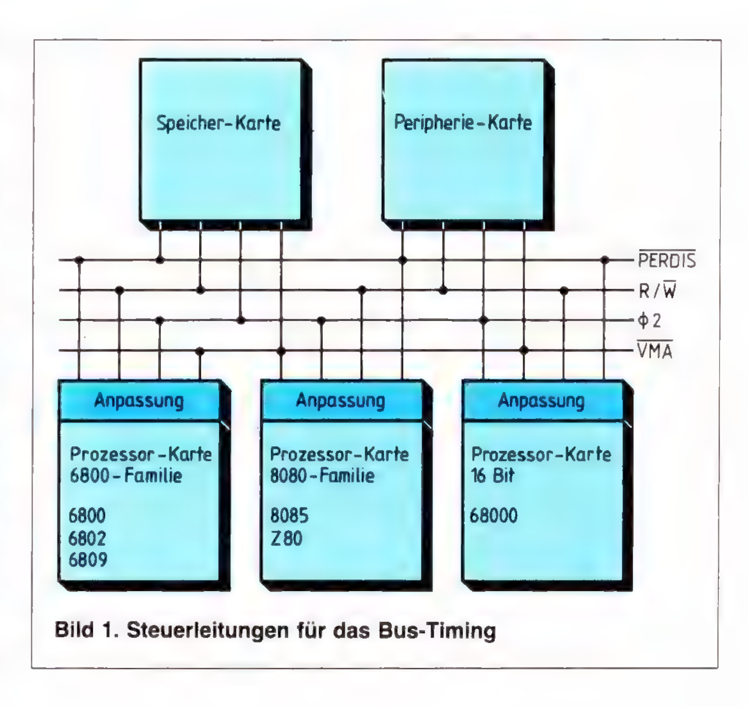


Bild 1. Steuerleitungen für das Bus-Timing

bzw. Peripherie verstanden. Die Prozessoren der 8080-Familie haben getrennte Steuerleitungen für Lesen und Schreiben und wickeln einen Speicherzyklus in mehreren Taktperioden ab. Die Peripheriebausteine können wahlweise wie Speicherbausteine oder auch getrennt angesprochen werden. Die Prozessoren der 6800-Familie haben eine gemeinsame Steuerleitung für Lesen und Schreiben sowie z. T. ein Sondersignal für gültige Speicheradressen (VMA) und benötigen einen Takt für einen Speicherzugriff. Die Peripherieadressen werden wie Speicheradressen behandelt.

Der 16-Bit-Prozessor 68000 kennt sowohl den asynchronen Speicher- als auch den synchronen Peripheriezugriff.

Bild 1 zeigt die Steuerleitungen für ein universelles Bussystem, das über Anpassungsschaltungen auf den jeweiligen Prozessorkarten den Speicher- und Peripheriekarten Steuersignale für die Datenübergabe zuführt. Diese entsprechen dem Verhalten der 6800-Prozessorfamilie. Die Leitung „PERDIS" dient als Synchronisationsanzeige beim Übergang vom asynchronen in den synchronen Betrieb.

3 Die Adressierung

Ein modernes und zukunftsorientiertes Bussystem muß sowohl für 8 Bit als auch für die neuen 16-Bit-Prozessoren ausgelegt sein. Das setzt einen 16-Bit-Datenbus und Adressierungsmöglichkeit über die 64 KByte der 8-Bit-Prozessoren voraus.

Bild 2. Steuerung der Peripherie

3.1 *Die Peripherie-Steuerung*

Die Peripherie wie z. B. Parallelschnittstellen oder Analog/Digital-Umsetzer benötigt nur einen relativ kleinen Adreßbereich im Vergleich zu den Speichern. Aus diesem Grund enthält das vorgestellte Bussystem eine Peripherie-Auswahlleitung. Ein Decoder-PROM auf den Prozessor-Karten legt die Adresse des Peripheriebereiches fest. *Bild 2* zeigt die Peripherieauswahlleitung „PER", die zur Auswahl der Peripheriekarten dient.

3.2 *Speichererweiterung über 64 KByte hinaus*

Folgende Gründe sprechen für eine Speichererweiterung:

- Moderne Betriebssysteme benötigen bereits annähernd 60 KByte, so daß dem Benutzer bei einem 64-K-System nur noch 4 KByte zur Verfügung ständen.
- Bei der Zusammenarbeit mit 16-Bit-Prozessoren müssen auch die 8-Bit-Prozessoren auf den erweiterten Adreßbereich dieser Prozessoren zugreifen können.
- Durch die erweiterte Speicherkapazität reduzieren sich die Zugriffe auf die Massenspeicher. Damit erhöht sich die Verarbeitungsgeschwindigkeit.
- Bei Betriebssystemen, die den Multi-User-Betrieb unterstützen, muß jedem Benutzer ein ausreichender Arbeitsspeicher zugewiesen werden.
- Grafikfähige Video-Umlaufspeicher benötigen eine große Arbeitskapazität und müssen sowohl von der Videosteuerung als auch vom System angesprochen werden können.

Bild 3 zeigt den Aufbau des Speicherbereiches aus max. 8 Blöcken zu je 64 KByte. Zu den 16 Adreßleitungen treten acht Blockauswahlleitungen, die von einer Block-Kontrollkarte gesteuert werden. Bei 16-Bit-Prozessoren bilden die unteren 16 Adreßleitungen den eigentlichen Adreßbus, während die oberen Leitungen zur Blockauswahl benutzt werden. Bei einem System mit Karten im Europaformat ist ein zentraler Decodierer trotz der acht Busleitungen aus Platzgründen günstiger als eine dezentrale Decodierung, die mit nur drei Leitungen auskäme.

3.3 *Gemeinsamer Speicher bei 8/16-Bit-Systemen*

Der Betrieb eines Prozessors allein oder mit einem anderen gleicher Bitbreite ist einfach. Die 8-Bit-Prozessoren arbeiten auf einen 8-Bit-Datenbus mit byteorganisierten Speicher- und Peripheriekarten. Die 16-Bit-Prozessoren arbeiten auf zwei 8-Bit-Datenbusse. Dabei erhebt sich die Frage, ob für 16-Bit-Systeme besondere Karten eingesetzt werden müssen oder ob mit den Karten der 8-Bit-Systeme gearbeitet werden kann.

Bild 4 zeigt eine Lösung, die universelle Speicher- und Peripheriekarten verwendet, mit denen sowohl Einzelbetrieb von Prozessoren einer Bitbreite als auch gemischter Betrieb von Prozessoren verschiedener Bitbreite möglich ist.

Die gezeigten RAM-Karten sind byteorganisiert; bei 16-Bit-Systemen wird eine Karte über Brücken auf den unteren und eine zweite auf den oberen Datenbus gelegt. Die EPROM-Karten werden über eine Kartensteuerung so umgeschaltet, daß sie sowohl auf den oberen als auch auf den unteren Datenbus als auch auf beide gleichzeitig arbeiten können. Die Peripheriekarten sind über Brükken umschaltbar, werden aber vorzugsweise über den unteren Datenbus betrieben. Die Steuerleitungen „UDS" und „LDS" steuern den oberen bzw. den unteren Datenbus. Sie sind bidirektional. Eine „WORD" genannte Steuerleitung teilt den Speicher- und Peripheriekarten mit, ob ein 8-Bit-Prozessor oder ein 16-Bit-Prozessor Daten überträgt. Bei einem gemischten 8-Bit-/16-Bit-Betrieb muß der 8-Bit-Prozessor auf Daten zugreifen können, die im Bereich des oberen Datenbus liegen. Die 8-Bit-Prozessorkarte legt also die WORD-Leitung konstant auf HIGH, und in diesem Fall erzeugen die Kartensteuerungen der Speicher die Steuersignale „UDS" und „LDS". Will z. B. eine Karte Daten aus dem Datenbereich übertragen, der am oberen Datenbus liegt, so bildet sie das Steuersignal „UDS", damit die Prozessorkarte die Daten auf dem oberen Datenweg annehmen kann. Karten mit 16-Bit-Prozessoren legen dagegen die WORD-Leitung konstant auf Low und erzeugen die Signale „UDS" und „LDS" selbst.

Die beiden RAM-Bausteine des Bildes 4 zeigen die Adreßverschiebung zwischen einem 8-Bit- und einem 16-Bit-Datenzugriff. Das Problem der unterschiedlichen Datenformate läßt sich leicht softwaremäßig lösen.

4 Die Busvergabe

Die Standardverfahren zum Auffrischen dynamischer Speicher und die Datenübertragung mit direktem Datenzugriff (DMA) sollen hier nicht weiter betrachtet werden. Bei der Parallelarbeit mehrerer Prozessoren muß sichergestellt werden, daß immer nur ein Prozessor die Kontrolle über den Adreßbus, Datenbus und die Steuerleitungen hat und daß sich alle anderen im Wartezustand befinden.

Bild 5 zeigt eine Schaltung zur Busvergabe, die auf den Prozessorkarten angeordnet ist. Ein Flipflop („Service-Request") speichert den Zustand des Prozessors (Prozessor aktiv/Prozessor wartet). Es kann über eine bestimmte Adresse und ein bestimmtes Datenbit über den Datenbus gesetzt und rückgesetzt werden.

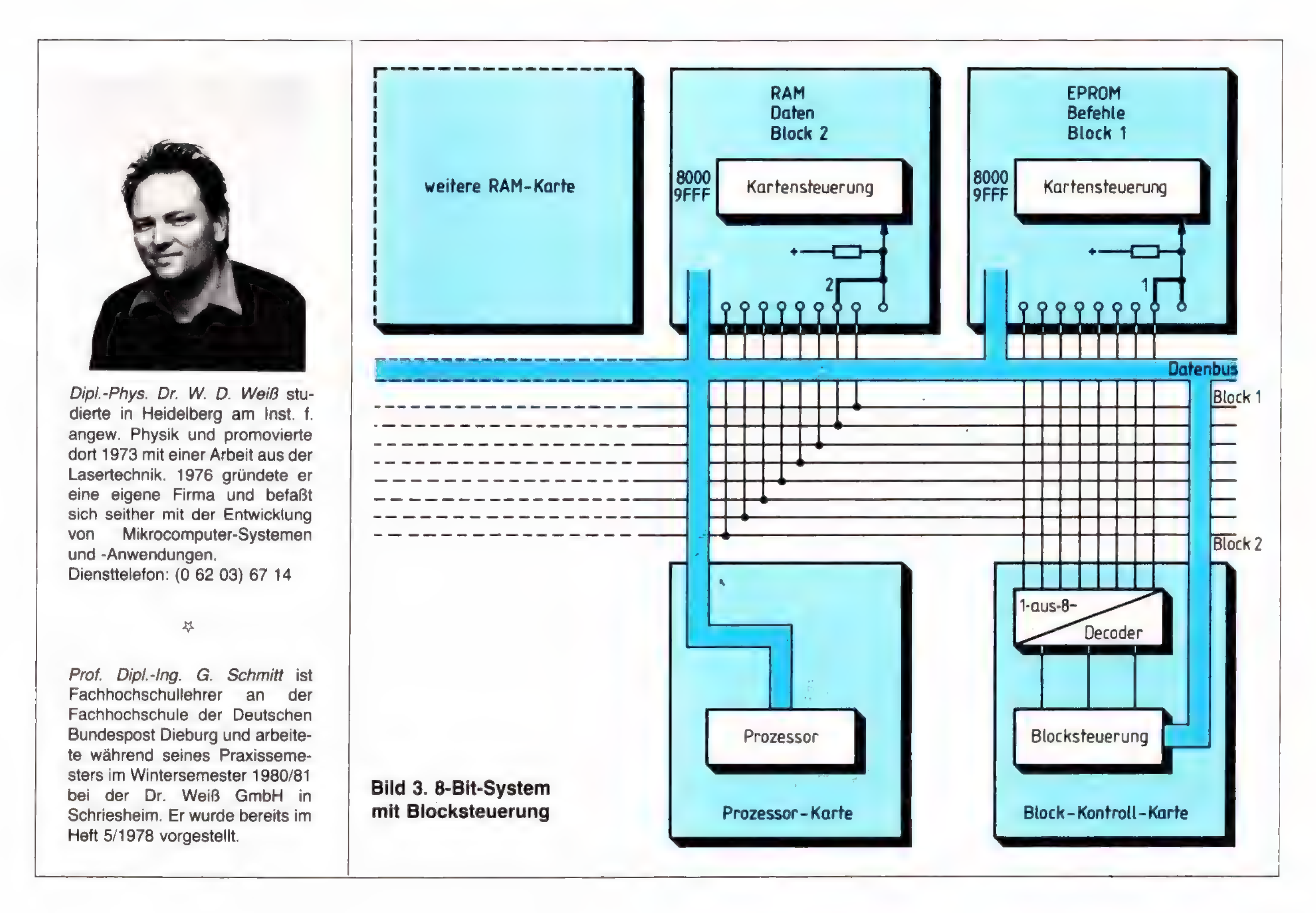


Bild 3. 8-Bit-System mit Blocksteuerung

Dipl.-Phys. Dr. W. D. Weiß studierte in Heidelberg am Inst. f. angew. Physik und promovierte dort 1973 mit einer Arbeit aus der Lasertechnik. 1976 gründete er eine eigene Firma und befaßt sich seither mit der Entwicklung von Mikrocomputer-Systemen und -Anwendungen.
Diensttelefon: (0 62 03) 67 14

☆

Prof. Dipl.-Ing. G. Schmitt ist Fachhochschullehrer an der Fachhochschule der Deutschen Bundespost Dieburg und arbeitete während seines Praxissemesters im Wintersemester 1980/81 bei der Dr. Weiß GmbH in Schriesheim. Er wurde bereits im Heft 5/1978 vorgestellt.

Bild 4. Gemischtes 8/16-Bit-System

Über Brücken lassen sich verschiedene Betriebsarten einstellen. Brücke I legt fest, ob sich der Prozessor an der Busvergabe beteiligt oder nicht. Brücke II ordnet dem Flipflop und damit der Karte ein Bit des Datenbus zu. Über dieses Bit kann das Flipflop gesetzt (Prozessor aktiv) oder rückgesetzt werden (Prozessor wartet). Damit können bei einem 8-Bit-Datenbus maximal acht Prozessoren parallel arbeiten. Die Prozessoren werden also über Befehle, die alle auf einer gemeinsamen Adresse liegenden Flipflops ansprechen, an den Bus geschaltet bzw. abgeschaltet. Brücke III legt fest, ob der Prozessor nach einem Reset die Kontrolle erhält oder nicht. Brücke IV dient zusammen mit den Steuereingängen „BUSRQIN“ und „BUSRQOUT“ zur Vergabe einer Priorität.

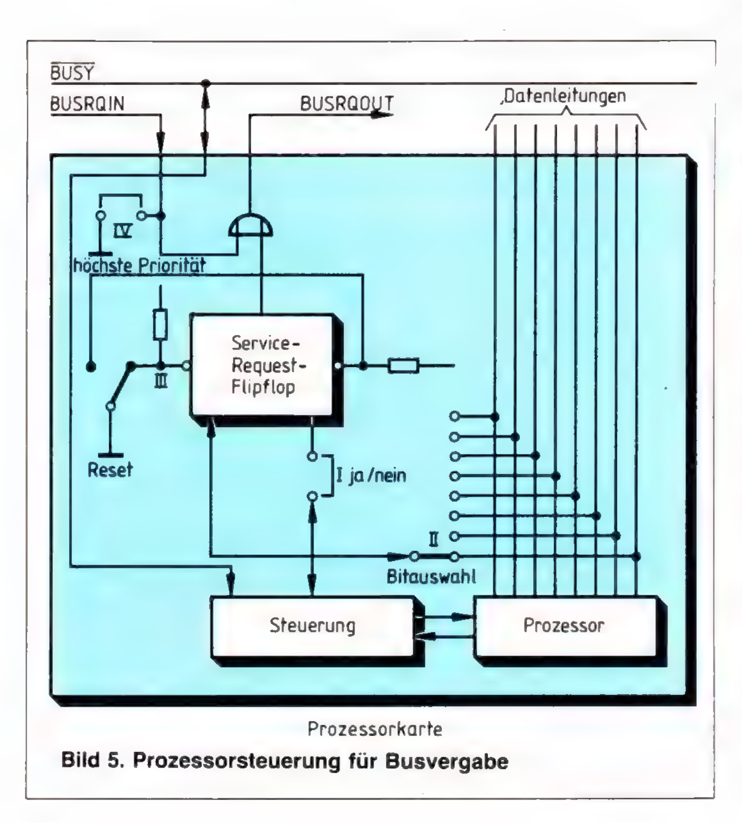


Bild 5. Prozessorsteuerung für Busvergabe

Der „neue“ Prozessor kann den Bus nur dann übernehmen, wenn der „alte“ seinen letzten Befehl vollständig ausgeführt und den Bus freigegeben hat. Dies wird über die Leitung „BUSY“ signalisiert. Hat der „neue“ Prozessor den Bus übernommen, so setzt er seinerseits die BUSY-Leitung aktiv.

Die Busvergabe kann nach dem Master-Verfahren, der Prioritätskette und der zentralen Verteilung erfolgen.

4.1 *Das Master-Verfahren*

Bei diesem Verfahren sind alle Prozessoren gleichberechtigt; jedoch muß durch Einstellen der Brücken dafür

gesorgt werden, daß bei einem Reset zunächst ein Prozessor die Kontrolle übernimmt. Dieser bestimmt durch Umschalten der Flipflops seinen Nachfolger. Die Busvergabe kann durch ein Betriebssystem nach verschiedenen Gesichtspunkten vorgenommen werden.

4.2 Die Prioritätskette

Bei der Prioritätskette („Daisy Chain") nach *Bild 6* legt die räumliche Anordnung der Prozessorkarten im Einschub eine Priorität innerhalb der Prozessoren fest.

Die Anschlüsse BUSRQIN und BUSRQOUT bilden auf dem Bus eine Leitung, die an jedem Steckplatz aufgetrennt ist. Liegt der BUSRQIN-Eingang einer Karte auf Low, so hat sie zur Zeit die höchste Priorität und kann die Buskontrolle übernehmen. Liegt der Eingang auf High, so kann sie die Kontrolle nicht übernehmen oder muß sie abgeben. Eine Karte, die die Buskontrolle übernommen hat oder übernehmen will, legt ihren BUSRQOUT-Ausgang auf High und sperrt damit alle dahinter liegenden Karten der Kette. Der BUSRQIN-Eingang der Karte mit der höchsten Priorität wird mit einer Brücke auf Low gelegt. Die Busvergabe kann wie unter Punkt 4.1 softwaremäßig durch ein Betriebssystem erfolgen, jedoch kann jetzt jede Karte entsprechend ihrer Priorität in die Busvergabe aktiv eingreifen.

4.3 Die zentrale Busvergabe

Trennt man die BUSRQIN/BUSRQOUT-Leitungen des Bildes 6 am Bus auf und führt die Anschlüsse getrennt einer Bus-Steuerkarte (Arbitrator) zu, so kann diese die Busvergabe zentral steuern. Jede Karte kann dabei Anforderungen über das Service-Request-Flipflop und die BUSRQOUT-Leitung absetzen; die Vergabe erfolgt jedoch vom Arbitrator über den BUSRQIN-Eingang. Der Arbitrator kann sowohl softwaremäßig mit einem eigenen Prozessor als auch mit schnellen Schaltungen hardwaremäßig aufgebaut sein.

5 Beispiel einer Busübergabe

Bild 7 zeigt als Anwendungsbeispiel eine Busübergabe zwischen dem 16-Bit-Prozessor 68000 und dem 8-Bit-Prozessor 6809.

Mit einem 68000-Entwicklungssystem bestehend aus einer Prozessor-Karte und einer Monitor-Karte ist ein Treiberprogramm für einen Analog/Digital-Umsetzer zu entwickeln. Nach einem Reset arbeitet zunächst das 68000-System. Die Editierung und Übersetzung des Treiberprogramms wird mit einem Editor und Cross-Assembler eines 6809-Systems vorgenommen. Für diese Entwicklungsphase gibt also das 68000-System die Kon-

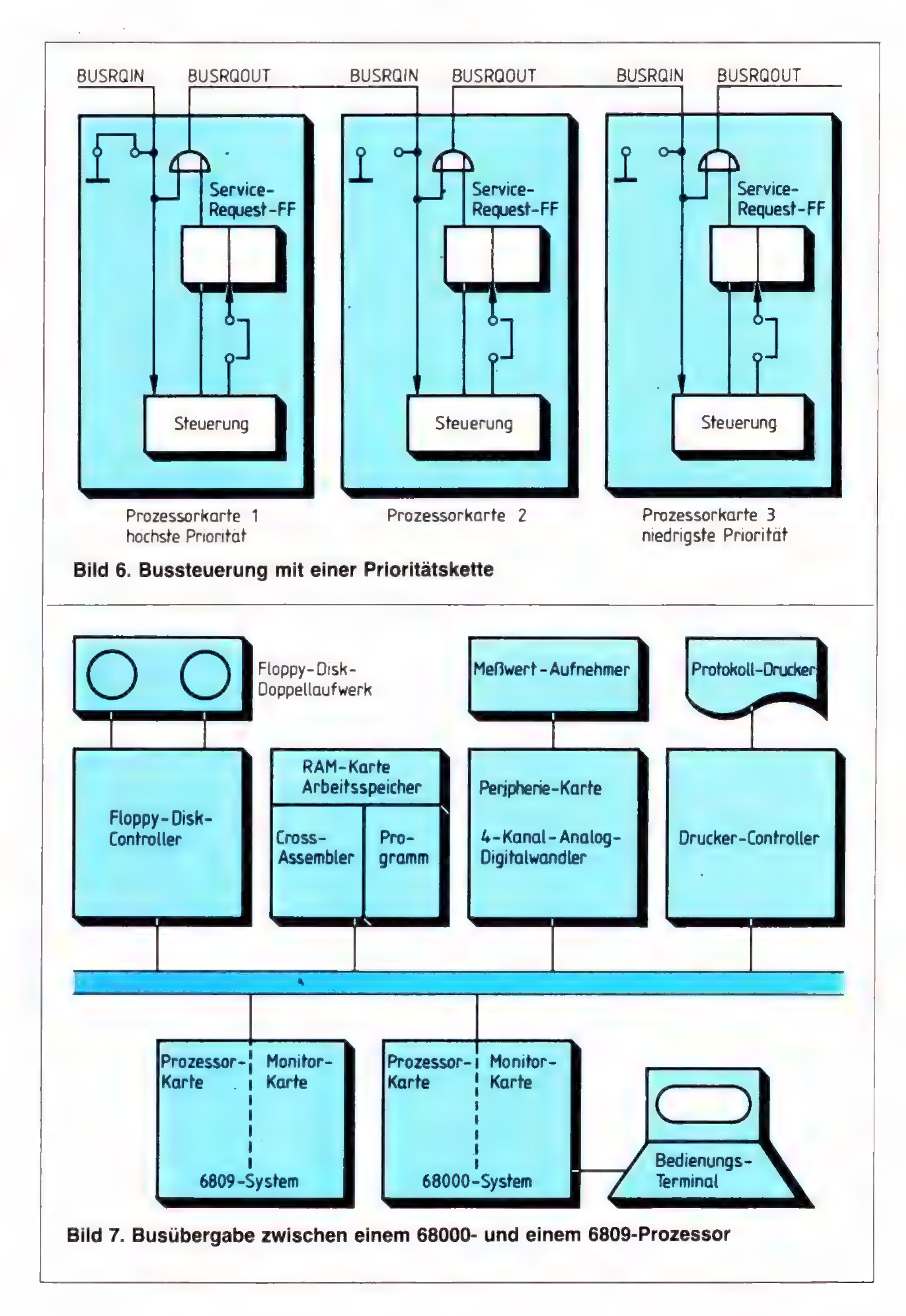


Bild 6. Bussteuerung mit einer Prioritätskette

Bild 7. Busübergabe zwischen einem 68000- und einem 6809-Prozessor

trolle an das 6809-System ab, das das fertige Maschinenprogramm im Format des 68000-Prozessors in den Arbeitsspeicher lädt und die Kontrolle wieder zurückgibt. Der Test des Programms erfolgt mit dem 68000-Entwicklungssystem.

6 Einsatzmöglichkeiten

Das vorliegende System besteht aus einem 96poligen Bussystem (MAKBUS) mit aufeinander abgestimmten Prozessor-, Betriebssystem-, Speicher-, Peripherie- und Geräteinterfacekarten im Europaformat. Infolge seiner Fähigkeit, verschiedene Prozessoren an einem Bus zu betreiben, ergeben sich folgende besondere Einsatzmöglichkeiten:

- Bei einem Übergang von einem 8-Bit- zu einem 16-Bit-System kann bestehende 8-Bit-Software verwendet werden.
- Bei einem Übergang von einer Prozessorfamilie zu einer anderen können ebenfalls bestehende Programme verwendet werden.
- Bei einem Wechsel von einer Prozessorfamilie zu anderen können die Peripherie- und Speicherkarten weiter benutzt werden. Damit vereinfacht sich die Lagerhaltung.
- Das System läßt sich leicht an Betriebssysteme anpassen, die in zunehmendem Maße von reinen Softwarehäusern angeboten werden.
- Da die Peripherie- und Speicherkarten für alle Prozessoren verwendbar sind, ist es nunmehr möglich, den für jede Anwendung günstigsten Prozessortyp einzusetzen, ohne nun an einen bestimmten Hersteller und dessen „Familie" gebunden zu sein.

Das vorliegende universelle Bussystem zeigt einen möglichen Ausweg aus der Softwarekrise, denn bisher mußte die Software für bestehende Hardware entwickelt werden; nun ist es möglich, die Hardware an die Software anzupassen. Dies verlangt jedoch einen Mikrocomputer-Entwickler, der bereit ist, mehr als einen Prozessor zu beherrschen.

Literatur

[1] *Engelhardt, H.* und *Feger, O.*: Multi-Mikrocomputersysteme. ELEKTRONIK 1978, H. 11, S. 49...53.

[2] *Althoff, H.-G.*: Multiprozessorbus-System für Europakarten. ELEKTRONIK 1980, H. 18, S. 57...62.

[3] *Stuhlmüller, P.*: Multiprozessorfähigkeit des Z8000 ermöglicht neue Rechnerkonzepte. ELEKTRONIK 1980, H. 19, S. 59...68.

[4] *Büdenbender, H.-W.*: Bussystem wird dezentral gesteuert. ELEKTRONIK 1981, H. 1, S. 81...85.

[5] Eine preiswerte Erweiterung des Standard-8-Bit-EUROMAK-Systems auf den Prozessor 68000. Firmenschrift Dr. Weiß GmbH, Schriesheim.

[6] EUROMAK-Systembeschreibung. Firmenschrift Dr. Weiß GmbH, Schriesheim.

Dipl.-Inf. Johann Geyer

32-Bit-Mikrocomputer besitzt neuartige Architektur

Nach mehr als fünfjähriger Entwicklungszeit stellte Intel den 32-Bit-Mikrocomputer iAPX 432 vor, der verglichen mit bisher bekannten Strukturen eine völlig andere Architektur aufweist. Der Grund dafür liegt in dem Ziel, das schon zu Beginn der Entwicklungsarbeiten vorgegeben wurde, nämlich unter Ausnutzung der VLSI-Technologie eine Rechnerarchitektur zu schaffen, die den Softwareerkenntnissen der siebziger Jahre entspricht, und die Grenzen zu überwinden, die bisher in bezug auf Leistung, Zuverlässigkeit sowie Funktionalität bestanden haben. Die Gesamtlösung umfaßt eine neuartige Architektur, realisiert in VLSI, Betriebssystemfunktionen in Silizium und eine Programmierumgebung für ADA.

1 Zielsetzung

Vor fast zehn Jahren stellte Intel mit dem 4004 den ersten Mikroprozessor vor. Seitdem werden Mikroprozessoren bei immer komplexeren Anwendungen eingesetzt. Dieses Anwachsen der Komplexität führte gleichzeitig, vor allem auf der Softwareseite, zu einem gravierenden Anstieg der Entwicklungskosten sowie zu einem Engpaß an geeigneten Systementwicklern und Programmierern.

Bei allen Rechnersystemen, ob Großrechner, Minicomputer oder Mikroprozessoren, sind heute Entwicklungskosten für Software höher als die für Hardware. In vielen Fällen ist es sogar bereits so, daß die auf das Einzelsystem umgelegten gesamten Softwarekosten höher sind als die Herstellungskosten der Hardware. Diese Tatsache ist vor allem dadurch zu erklären, daß in der Regel 70 % der Softwarekosten erst nach dem eigentlichen Entwicklungsabschluß entstehen.

Diese oft auch als „Softwarekrise" bezeichnete Situation war bereits Mitte der siebziger Jahre erkennbar; insbesondere durch Einführung der „strukturierten Programmierung" wurde versucht, hier Verbesserungen zu erreichen. Eine 1975 bei Intel in den USA etablierte Gruppe von Entwicklern hatte das Ziel, dieses Problem von einer anderen Seite her anzugehen. Beim Start des Projekts war gerade der Mikroprozessor 8080 auf dem Markt, der 5500 Bauelemente in einem Chip enthält. Seinerzeit besagte das „Gesetz" von *Gordon Moore*, einem Gründer von Intel, daß sich die Anzahl der Bauelemente pro Chip alle ein bis zwei Jahre verdoppeln würde. Die Entwicklungsgruppe sah sich daher mit der Frage konfrontiert, was 1980 mit 100 000 Einzelbauelementen auf einem Chip geschehen sollte.

Die Integration von 16 Mikroprozessoren des Typs 8080 zu einem Bauelement wäre zwar alleine vom Leistungsgesichtspunkt aus interessant gewesen, jedoch hätte eine solche Feldrechnerarchitektur in keiner Weise Softwareprobleme gelöst, sondern im Gegenteil sogar neue geschaffen.

Die Untersuchung komplexer Anwendungen im oberen Leistungsbereich von Mikroprozessoren und des Einsatzes von Minicomputern und Großrechnern ergab mehrere Gemeinsamkeiten. Daraus leiteten sich die Entwurfsziele für eine neue Architektur ab:

- Diese Anwendungen sind sehr softwareintensiv; dabei laufen Programme für unterschiedliche Aufgaben parallel zueinander ab und arbeiten teilweise sogar mit gemeinsamen Daten. Als Beispiel dafür sei hier das Zusammenwirken von Wareneingang, Buchhaltung, Lagerbestandsführung und Fertigungsvorbereitung in einem Industriebetrieb genannt. Für diese softwareintensiven Anwendungen müßte eine neue Rechnerarchitektur die Produktivität der Programmierer in allen Phasen unterstützen, um die Kosten der Erstellung, der Wartung und der Weiterentwicklung von Programmen zu senken.
- Diese Anwendungen erfordern eine hohe Verarbeitungsleistung. In dieser Leistung muß auch die Unterstützung von Datentypen und Laufzeitsystemen höherer Programmiersprachen sowie moderner und sicherer Betriebssysteme eingeschlossen sein. Im Maximalausbau müßte das zu entwickelnde System über der Leistung der schnellsten Minicomputer liegen und die von mittleren Großrechnern erreichen.
- Gleiche Anwendungen erfordern oft stark unterschiedliche Leistungen, z. B. Datenbankabfragen

von 5 oder 30 Terminals oder Vermittlungsrechner für 1000 oder 10 000 Telefonanschlüsse. Die Erweiterbarkeit der Leistung – sogar im Feld – um einen sehr großen Faktor ohne jegliche Softwareänderung, war von Anfang an Ziel beim Entwurf des iAPX 432.

- Diese Anwendungen stellen hohe Anforderungen an die Zuverlässigkeit von Hard- und Software. Für Vermittlungsrechner der Deutschen Bundespost wird beispielsweise gefordert, daß sie innerhalb von 30 Jahren nur 2 Stunden ausfallen dürfen. Entwurfsziel der 432-Architektur war es daher, den Aufbau von ausfallsicheren Systemen zu ermöglichen und die Auswirkungen von Softwarefehlern so klein und so lokal wie möglich zu halten.

Während der Projektentwicklung erkannte man bereits sehr früh, daß es zum Erreichen der oben gezeigten Ziele einer neuen Rechnerarchitektur bedurfte und daß diese keinerlei Kompatibilitätsanforderungen unterworfen sein durfte. Im folgenden wird das Ergebnis des Entwicklungsprojekts, die Architektur des iAPX 432 und zwar sowohl die Hardware als auch die Software beschrieben.

2 Systemarchitektur

Um die Leistung eines 432-Systems innerhalb einer großen Bandbreite erweitern zu können, wurde dessen Architekur als Mehrprozessorsystem konzipiert. Minicomputer- und Großrechnerhersteller erreichen eine gewisse Variabilität in der Leistung oft durch verschiedene Implementierungen der gleichen Architektur, z. B. einmal mit Bit-Slices, ein anderes Mal mit ECL-Bausteinen oder durch mehr oder weniger starke Mikroprogrammierung. Diese Methoden fordern jedoch wesentlich höhere Entwicklungskosten und bieten zudem nicht die Möglichkeit der Leistungserweiterung im Feld. Aus dem Gesichtspunkt der Zuverlässigkeit ist hingegen ein Mehrprozessorkonzept überlegen, da hier beim Ausfall einer CPU nur die Verarbeitungsleistung zwar etwas sinkt, nicht aber das Gesamtsystem stillsteht.

Die 432-Architektur implementiert vollständig das Konzept der „transparenten Mehrfachverarbeitung" (Multiprocessing), d. h. daß die Anzahl der Prozessoren in einem 432-System ohne jegliche Änderung der Software erhöht oder vermindert werden kann. Es können sogar während des Betriebs Prozessoren angehalten oder hinzugefügt werden, ohne auch nur einen Befehl im Betriebssystem oder Anwenderprogramm zu verändern.

Diese Architektur bietet aber nicht nur die Möglichkeit, mehrere gleichartige Prozessoren miteinander zu verbinden, sondern auch unterschiedliche Prozesoren zusammenarbeiten zu lassen.

Als erste Realisierungen wurden jetzt der *iAPX 432 General Data Processor* (GDP) für die Verarbeitung von Programmen und der *iAPX 432 Interface Processor* (IP) für die Ein-/Ausgabe vorgestellt.

Ein grundsätzliches Problem bei Mehrprozessorsystemen stellt die Definition der Verbindung zwischen den einzelnen Prozessoren und dem Arbeitsspeicher dar. Statt diese Verbindung, also den Bus, festzulegen, spezifiziert die 432-Architektur nur, wie die einzelnen Prozessoren an den Bus angeschlossen werden, nämlich über das Paketbus-Protokoll.

2.1 Paketbus-Protokoll

Die Kommunikation zwischen den Prozessoren und dem Speicher basiert auf der Grundlage von Paketen. Im Falle eines Lesezugriffes schickt ein Prozessor eine Anforderung an den Arbeitsspeicher, der nach einiger Zeit die gelesene Information, ebenfalls als Paket, zurückschickt (*Bild 1*). Bei einem Schreibzugriff überträgt der Prozessor die Anforderungen zusammen mit den Daten in einem Paket zum Speicher. Eine Anforderung spezifiziert neben der physikalischen Speicheradresse auch, ob der Zugriff lesend, schreibend oder modifizierend (*Read-Modify-Write*) erfolgt, sowie die Anzahl der Bytes, auf die sich der Zugriff erstreckt, die variabel zwischen eins und sechzehn liegen kann.

Das Paketbus-Protokoll hat den wesentlichen Vorteil, daß es die Busbelegung vermindert, weil dieser nach der Übertragung eines Pakets sofort wieder frei ist, zwischen den zwei Paketen eines Lesezugriffs also andere Pakete über den Bus gehen können, und weil

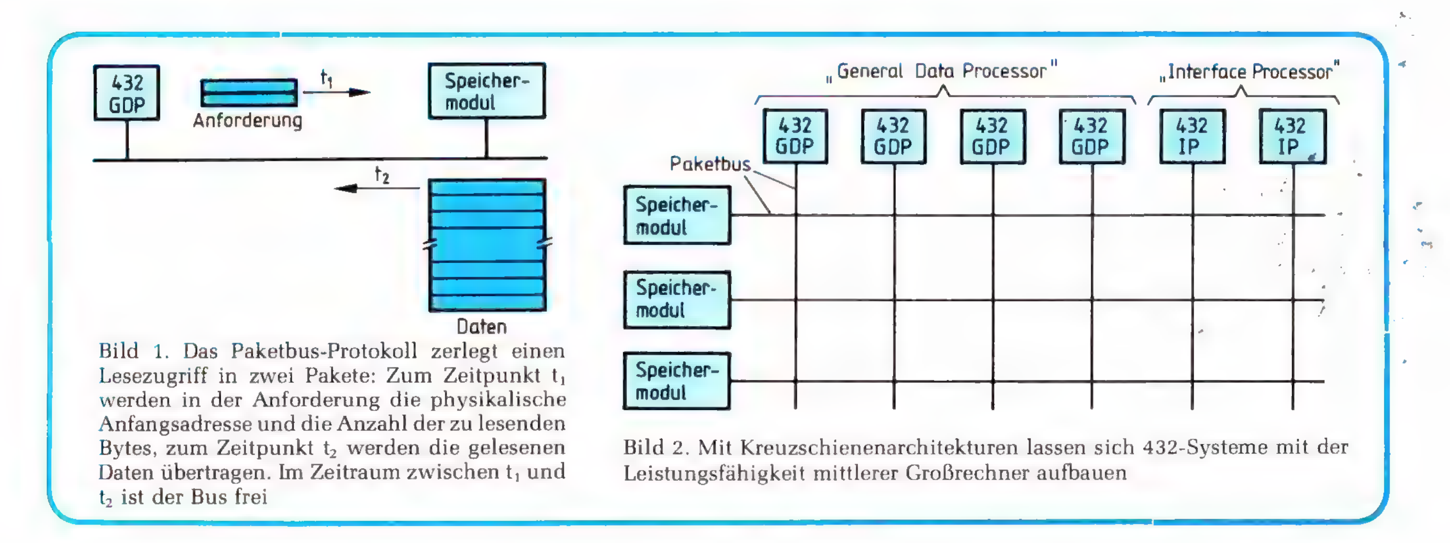


Bild 1. Das Paketbus-Protokoll zerlegt einen Lesezugriff in zwei Pakete: Zum Zeitpunkt t_1 werden in der Anforderung die physikalische Anfangsadresse und die Anzahl der zu lesenden Bytes, zum Zeitpunkt t_2 werden die gelesenen Daten übertragen. Im Zeitraum zwischen t_1 und t_2 ist der Bus frei

Bild 2. Mit Kreuzschienenarchitekturen lassen sich 432-Systeme mit der Leistungsfähigkeit mittlerer Großrechner aufbauen

selbst bei längeren Datenelementen die Adresse nur einmal angegeben werden muß. Das Paketbus-Protokoll erlaubt weiterhin, bis zu 16 Byte in einem Read-Modify-Write-Zyklus zu verändern sowie die direkte Übertragung von Nachrichten zwischen den einzelnen Prozessoren, um beispielsweise einen von ihnen anzuhalten.

2.2 Busstruktur

Aufbauend auf diesem Paketbus-Protokoll lassen sich unterschiedliche Architekturen für die Verbindung zwischen den Prozessoren und dem Arbeitsspeicher implementieren. Am einfachsten ist es wohl, den am Prozessor anliegenden Paketbus durch diskrete Logik zu entmultiplexen und die nun getrennten Adreß- und Datenleitungen in konventioneller Weise an den Speichermodul anzulegen. Bis zu vier Prozessoren können ohne größere gegenseitige Beeinträchtigung an diesen Bus angeschlossen werden.

Ein völlig anderer, 32 Bit breiter Bus unterstützt im Platinensystem 432/200 bis zu sechs Prozessoren. 432-Systeme maximaler Leistung können durch Kreuzschienenarchitekturen aufgebaut werden, in denen parallele Busse mehreren Prozessoren den gleichzeitigen Zugriff auf die einzelnen Speichermoduln erlauben (*Bild 2*).

2.3 Ein-/Ausgabe

Ein wichtiges Merkmal der 432-Architektur ist es, die Verarbeitung der Programme von der Ein-/Ausgabe zu trennen. Diese erfolgt in eigenständigen E/A-Untersystemem, die die Aufgabe der Gerätеansteuerung, der Interruptbehandlung und des Datentransfers übernehmen. Diese E/A-Untersysteme bestehen in der Regel aus einem Standardmikroprozessor wie dem 8086, der über einen eigenen Arbeitsspeicher verfügt und die Peripheriebausteine bedient (*Bild 3*).

Die Verbindung zwischen dem System und dem Mikroprozessor erfolgt über den *432 Interface Processor* (IP). Der IP ermöglicht es dem externen Mikroprozessor auf den 432-Arbeitsspeicher zuzugreifen, indem er die physikalischen Adressen des Mikroprozessorsystems auf den 432-Adreßraum abbildet. Dazu kann der Mikroprozessor im IP sogenannte Fenster definieren, die bestimmte Adreßbereiche des Untersystems einzelnen Objekten im 432-Arbeitsspeicher zuordnen. Eines dieser Fenster ist für Blocktransfer optimiert, damit die Daten von oder zu schnellen Peripheriegeräten direkt von einem DMA-Controller oder E/A-Prozessor 8089 durch das Fenster übrtragen werden können. Bei allen Zugriffen auf den 432-Speicher arbeitet der IP mit derselben objektorientierten Adressierungstechnik wie der GDP und überprüft auch die Zugriffsrechte in gleicher Weise.

Alle Interaktionen zwischen Programmen eines 432-Systems basiert auf dem Austausch von Nachrichten im Arbeitsspeicher; es gibt hier keine Interrupts. Um auch den Mikroprozessor des E/A-Untersystems in dieses Konzept einzubinden, kann dieser den IP

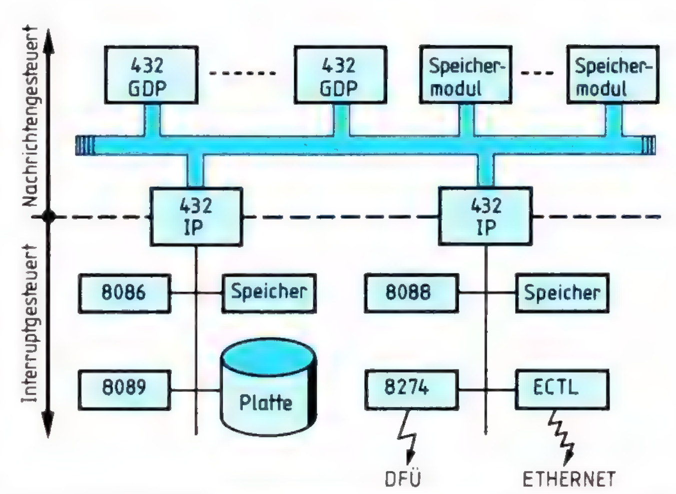


Bild 3. Standardmikroprozessoren übernehmen die Ein-/Ausgabe in einem 432-System. Die Verbindung zwischen der nachrichtengesteuerten 432-Architektur und dem interruptgetriebenen E/A-Untersystem übernimmt der „432 Interface Processor"

durch Kommandos beauftragen, für ihn Nachrichten zu empfangen oder zu senden. Die Ausführung eines solchen Kommandos meldet der IP durch einen Interrupt an den Mikroprozessor zurück. Eine empfangene Nachricht kann der Mikroprozessor anschließend über ein Fenster des IP adressieren.

Der IP bietet dem externen Mikroprozessor eine Reihe von Kommandos an, die den Instruktionen und Hardwarefunktionen des sogenannten „Betriebssystems in Silizium" des GDP entsprechen, und „erweitert" somit den Befehlssatz des Mikroprozessors um diese Eigenschaften.

Die Anzahl der Interfaceprozessoren in einem 432-System ist ebenso variierbar wie die der GDPs, und damit sind sowohl Verarbeitungs- als auch Ein-/Ausgabeleistung optimal an die einzelne Anwendung anpaßbar. Das Konzept der eigenständigen E/A-Untersysteme bietet weiter die Möglichkeit, bereits heute bestehende Software unverändert weiterzuverwenden, weil sie ja auf einem geeigneten Untersystem laufen kann.

2.4 VLSI-Implementierung

Die Prozessoren der 432-Familie werden in HMOS-Technologie gefertigt. Der 432 GDP enthält insgesamt 160 000 Transistorfunktionen, die auf zwei Bausteine verteilt sind. Der 43201 mit seinen etwa 100 000 Transistorfunktionen enthält ein 4-K x 16-Bit-Mikrocode-ROM und setzt die einzelnen Maschinenbefehle in Mikrobefehle um. Der 43202 führt diese aus und übernimmt die gesamte Adressierung (*Bild 4*).

Der 432 Interface Processor entspricht etwa 60 000 Transistorfunktionen und enthält ein 2-K x 16-Bit-Mikrocode-ROM, bei dem mit einer neuen Technologie, die auch beim Numerik-Prozessor 8087 angewandt wird, in einer Zelle zwei Bits gespeichert werden. Bedingt durch seine Funktion hat er einerseits die Anschlüsse zum 432-Paketbus, andererseits aber auch die Signale für die Mikroprozessorseite (*Bild 5*).

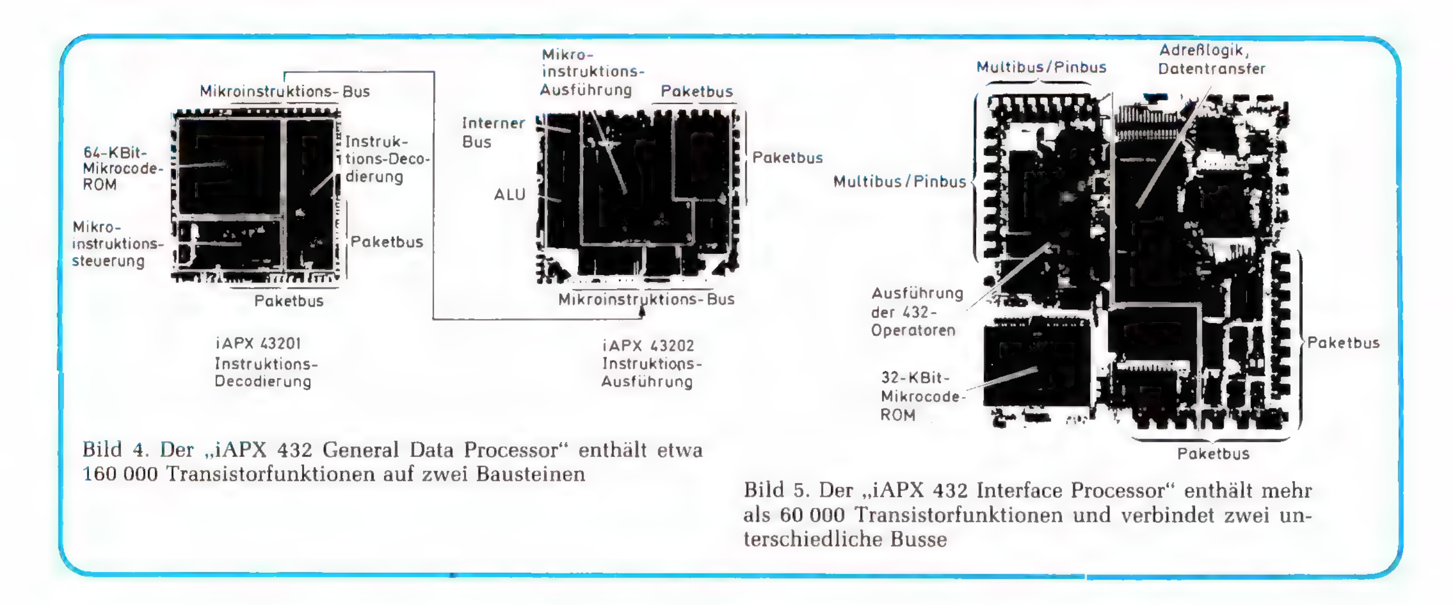


Bild 4. Der „iAPX 432 General Data Processor" enthält etwa 160 000 Transistorfunktionen auf zwei Bausteinen

Bild 5. Der „iAPX 432 Interface Processor" enthält mehr als 60 000 Transistorfunktionen und verbindet zwei unterschiedliche Busse

Bei beiden Prozessoren fällt die sehr regelmäßige Entwurfsstruktur und die starke Mikroprogrammierung auf. Nur dadurch ist es möglich, Bausteine solcher Komplexität zu entwerfen und zu testen.

Für die 432-Bausteine mußte auch ein neues Gehäuse entwickelt werden, das 64polige Quad-Inline-Package (QUIP). Es besteht aus einer Fassung, in dem ein Keramikträger, der den eigentlichen Chip enthält, durch eine Metallklammer festgehalten wird (*Bild 6*). Dieses Gehäuse bietet neben seiner Kompaktheit unter anderem den Vorteil, daß die Wärmeabgabe durch die Metallklammer sehr gut ist, wodurch der darunter befindliche Chip relativ kühl bleibt [1].

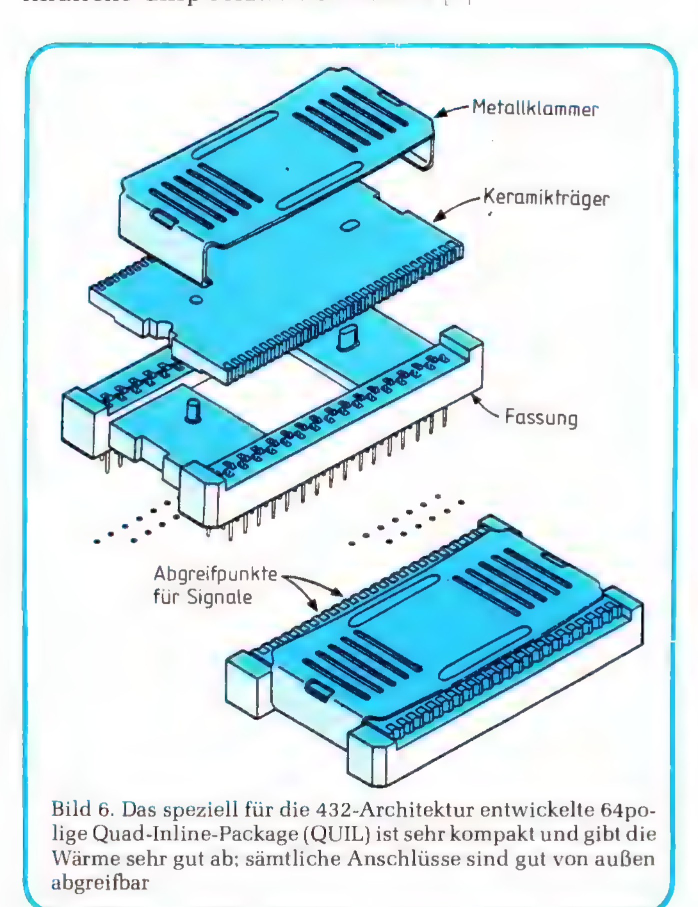


Bild 6. Das speziell für die 432-Architektur entwickelte 64polige Quad-Inline-Package (QUIL) ist sehr kompakt und gibt die Wärme sehr gut ab; sämtliche Anschlüsse sind gut von außen abgreifbar

2.5 Funktionsüberwachung durch Redundanztest

Um den Aufbau besonders sicherer Systeme zu ermöglichen, bieten alle Bauelemente dieses Systems die Möglichkeit des sogenannten *Functional Redundancy Checking* (FRC). Dabei werden bei zwei gleichen Bauteilen alle Anschlüsse direkt miteinander verbunden. Ein Baustein wird durch Anlegen von 5 V an einen speziellen Anschluß „*Master*", der andere durch Verbinden des entsprechenden Anschlusses mit Masse als „*Checker*" definiert. Beide Bausteine laufen völlig taktsynchron und erhalten die gleiche Information auf ihren Eingangsleitungen. Der Checker überprüft bei jedem Takt was der Master an seinen Ausgängen erzeugt und vergleicht es mit dem, was er selbst intern generiert hat. Sobald die beiden Bausteine bei irgendeinem Takt nicht übereinstimmen, erzeugt der Checker ein Signal, das den Master augenblicklich blockiert. Gleichzeitig kann es in der Außenwelt als Fehlersignal verwendet werden (*Bild 7*).

Durch das sofortige Anhalten der beiden Bausteine wird verhindert, daß falsche Information auf den Bus und damit in den Speicher gelangt. Mit dieser ohne jeden zusätzlichen Hardwareaufwand möglichen Methode können Systeme mit verschiedenem Zuverlässigkeitsgrad aufgebaut werden.

3 Softwarestruktur

Bereits in der Anfangsphase dieses Projekts standen alle daran Beteiligten vor dem Problem, die vielen Architekturmerkmale durch ein einheitliches Konzept zu verbinden, denn nur ein einheitliches Konzept würde das Überprüfen, Erweitern und leichte Verstehen der komplexen Eigenschaften der 432-Architektur ermöglichen. Nach langen Untersuchungen entschied man sich, die erst Anfang der siebziger Jahre entwickelte Idee des „objektorientierten Entwurfs" als konzeptionelle Grundlage für den iAPX 432 heranzuziehen. Bei diesem „objektorientierten Entwurf" wird versucht, durch Modularität die für das Verständnis überflüssige Information zu verbergen und Daten zu abstrahieren [2, 3].

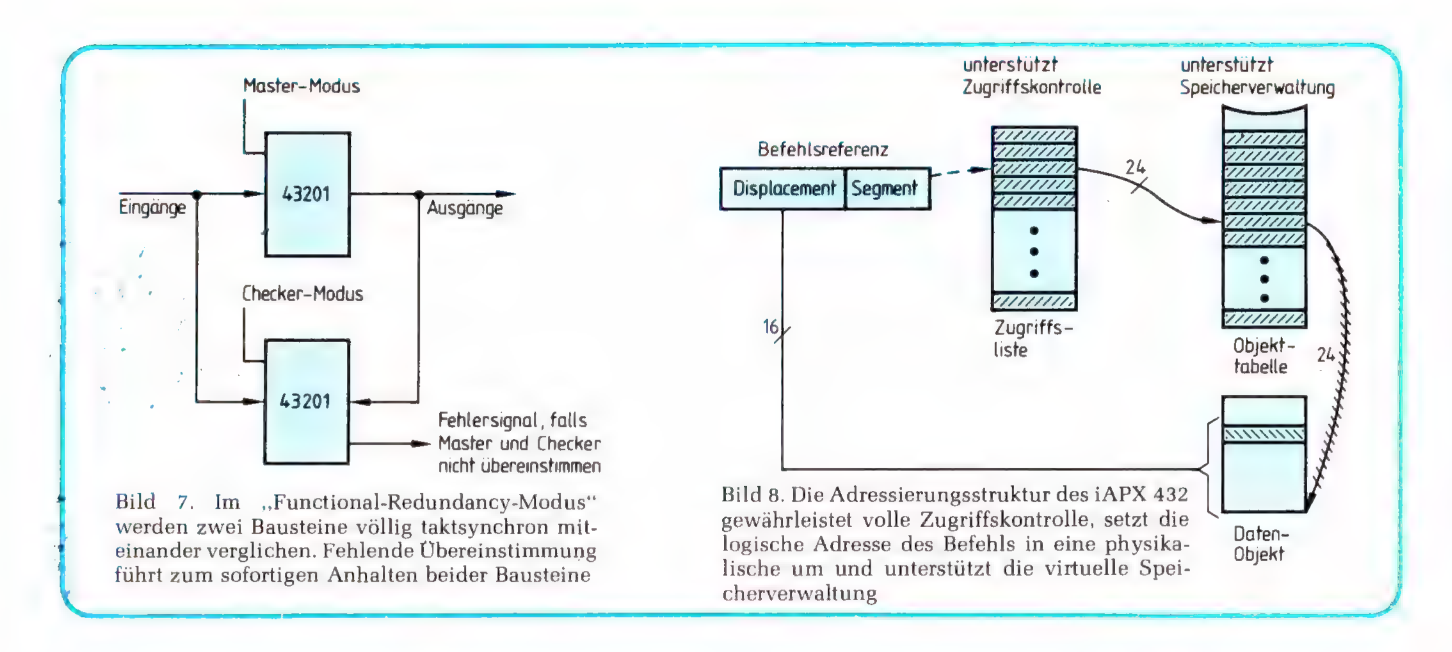


Bild 7. Im „Functional-Redundancy-Modus" werden zwei Bausteine völlig taktsynchron miteinander verglichen. Fehlende Übereinstimmung führt zum sofortigen Anhalten beider Bausteine

Bild 8. Die Adressierungsstruktur des iAPX 432 gewährleistet volle Zugriffskontrolle, setzt die logische Adresse des Befehls in eine physikalische um und unterstützt die virtuelle Speicherverwaltung

Deswegen definieren auch Objekte alle wesentlichen Eigenschaften des iAPX 432; er hat eine objektorientierte Architektur:

- Adressierung und Zugriffsschutz:
  Objekte verwalten den logischen und physikalischen Adreßraum und bilden die Grundlage für die Schutzmechanismen des iAPX 432.
- Allgemeine Verarbeitung:
  Datenobjekte unterstützen die allgemeine Verarbeitung durch eine große Anzahl von primitiven Datentypen.
- Laufzeituntersützung von Sprachen:
  Domain- und Contextobjekte unterstützen die Laufzeitumgebung von ADA, aber auch prozedurale Sprachen wie COBOL und FORTRAN.
- Betriebssystem in Silizium:
  Prozessor-, Port- und Prozeßobjekte steuern die Zuteilung der aktiven Betriebsmittel; Storage-Resource-Objekte definieren die Zuteilung des logischen und physikalischen Adreßraumes.

3.1 *Adressierung und Zugriffsschutz*

Die gesamte Adressierung innerhalb der 432-Architektur basiert auf dem Objektkonzept, wobei ein Objekt aus einem oder mehreren Segmenten besteht. Ein Segment ist ein zusammenhängender Speicherbereich mit einer Länge zwischen 1 Byte und 64 KByte. Jedes Objekt wird durch genau einen Objektdeskriptor beschrieben und kann auch nur über diesen adressiert werden.

Ein Objektdeskriptor enthält die physikalische Basisadresse eines Objekts, seinen Typ sowie seine Länge. Damit kann bei jedem Zugriff überprüft werden, ob ein Befehl auf Daten außerhalb des Objekts zugreift. Mehrere Bit unterstützen virtuelle Speicherverwaltung, eines sagt z. B., ob das Objekt sich überhaupt im Arbeitsspeicher befindet. Um auf die einzelnen Objektdeskriptoren in geordneter Weise zugreifen zu können, sind sie in einer zentralen Objekttabelle zusammengefaßt, die wiederum selbst ein Objekt ist; sie enthält somit auch ihren eigenen Objektdeskriptor.

Zum Zugriff auf ein Objekt benötigt ein Programm einen speziellen Zugriffsdeskriptor, der auf den entsprechenden Objektdeskriptor zeigt; dazu enthält der Zugriffsdeskriptor einen Index in die Objekttabelle. Außerdem definiert der Zugriffsdeskriptor die Zugriffsrechte des Programms auf ein Objekt. Damit kann erreicht werden, daß zwei Programme, deren jeweilige Zugriffsdeskriptoren auf denselben Objektdeskriptor, und damit auf dasselbe Objekt, zeigen, mit unterschiedlichen Rechten darauf arbeiten können. Sämtliche Zugriffsdeskriptoren eines oder mehrerer Unterprogramme sind wiederum in einer Zugriffsliste zusammengefaßt (*Bild 8*); sie definiert die Zugriffsumgebung dieses Programmteils (Context). Dieses Konzept bildet die Grundlage für die „Need-to-know-" oder „Capability-orientierte" Adressierung [4, 5] der 432-Architektur:

- Ein Programm kann nur auf ein Objekt zugreifen, wenn es einen Zugriffsdeskriptor darauf hat.
- Ein Programm kann nur mit den im Zugriffsdeskriptor angegebenen Rechten auf das Objekt zugreifen. Zugriffslisten sind ebenfalls von den Prozessoren erkannte Objekte. Es kann damit erreicht werden, daß nur spezielle Befehle auf ihnen operieren; Zugriffsdeskriptoren können also nicht wie normale Daten manipuliert und damit auch nicht gefälscht werden.

Diese Adressierungsstruktur ermöglicht es dem iAPX 432 bei einem physikalischen Adreßraum von 16 Megabyte einen logischen Adreßraum von 2^{40} Byte, also tausend Gigabyte anzubieten.

Damit die in Bild 8 gezeigte Adreßumsetzung über die im Arbeitsspeicher liegenden Tabellen nicht zu hohen Geschwindigkeitseinbußen führt, werden die Objektdeskriptoren der zuletzt benutzten Objekte in Cacheregistern auf dem Prozessor gehalten. Beim Zugriff auf ein nicht im Cache beschriebenes Objekt wird der zugehörige Deskriptor automatisch auf den Prozessor geladen. Im allgemeinen garantiert die Lokalität von Programmen eine gute Trefferrate für den Cache.

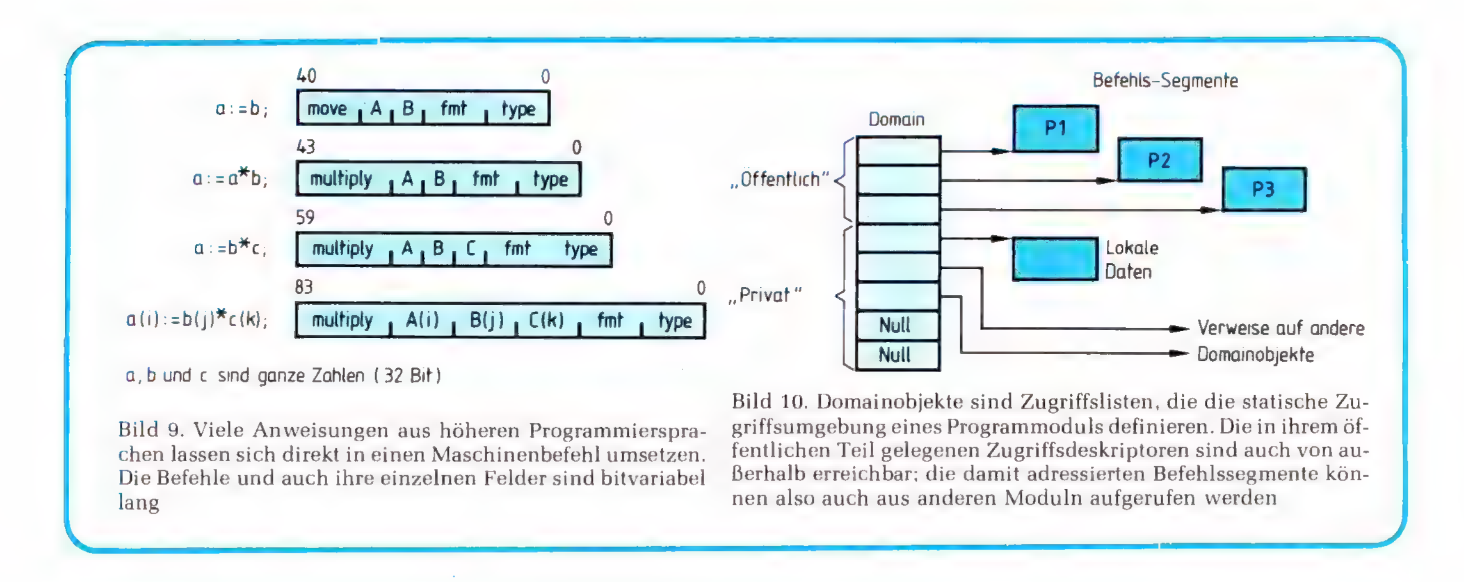


Bild 9. Viele Anweisungen aus höheren Programmiersprachen lassen sich direkt in einen Maschinenbefehl umsetzen. Die Befehle und auch ihre einzelnen Felder sind bitvariabel lang

Bild 10. Domainobjekte sind Zugriffslisten, die die statische Zugriffsumgebung eines Programmoduls definieren. Die in ihrem öffentlichen Teil gelegenen Zugriffsdeskriptoren sind auch von außerhalb erreichbar; die damit adressierten Befehlssegmente können also auch aus anderen Moduln aufgerufen werden

3.2 Allgemeine Verarbeitung

Die allgemeine Verarbeitung im iAPX 432 wird durch Datenobjekte unterstützt. Sie bestehen aus genau einem Segment und enthalten primitive Datentypen. Die Referenz für ein einzelnes Datenlement wird innerhalb eines Befehls durch Angabe eines Segmentselektors und des Displacements gebildet (Bild 8). Der iAPX 432 GDP verarbeitet als primitive Datentypen:

- Buchstaben und Boolesche Größen (8 Bit)
- ganze Zahlen ohne Vorzeichen (8, 16 und 32 Bit)
- ganze Zahlen mit Vorzeichen (16 und 32 Bit)
- Gleitpunktzahlen (32, 64 und 80 Bit).

Alle Formate sind kompatibel zum vorgeschlagenen IEEE-Standard, damit also auch zum Numerikprozessor 8087 [6]. Die Befehlsformate des iAPX 432 GDP sind so ausgelegt, daß sie die Codegenerierung in Übersetzern vereinfachen und die Programmgröße reduzieren. Alle Instruktionen sind bitvariabel in der Länge, um eine optimale Codierung entsprechend ihrer statistischen Häufigkeit zu erreichen. Der Befehlszähler des Prozessors zeigt somit auch auf Bitpositionen innerhalb eines Befehlsobjekts. Die einzelnen Befehle können zwischen null und drei Referenzen haben, wobei die Adressierungsarten so ausgelegt sind, daß sie alle Datenstrukturen höherer Programmiersprachen direkt unterstützen, nämlich Skalare, statische und dynamische Vektoren und Records. Auf die einzelnen Operanden wird entweder über Speicheradressen zugegriffen oder sie stehen im Stack. Um speziell die Erstellung von Übersetzern zu vereinfachen, gibt es keine von der Software aus ansprechbaren Register.

Dieses Befehlsformat garantiert, daß viele Anweisungen aus höheren Programmiersprachen auch in einen einzigen Maschinenbefehl umgesetzt werden können. *Bild 9* gibt hier einige Beispiele, die auch die Länge der Instruktionen in Bit enthalten.

Genauso wie Datenobjekte die allgemeine Verarbeitung von primitiven Datentypen unterstützen, erlauben komplexere Objekte die Ausführung höherliegender Funktionen. Die Hardware erkennt dabei im Objektdeskriptor eines solchen „Systemobjekts“ dessen genauen Typ und kann dann darauf Aktionen ausführen, die in konventionellen Architekturen größeren Unterprogrammen entsprechen.

3.3 Laufzeitunterstützung von Sprachen

Domain- und Contextobjekte dienen in erster Linie der Laufzeitunterstützung von höheren Programmiersprachen. Dabei definiert ein Domainobjekt die statische Zugriffsumgebung eines Programmteiles; es enthält Verweise, d. h. Zugriffsdeskriptoren auf alle Unterprogramme des Moduls und seine statisch angelegten Daten (*Bild 10*). Ein modular aufgebautes Programm wird durch ein Netz miteinander verbundener Domainobjekte repräsentiert.

Der „öffentliche“ Teil eines Domainobjekts legt die Schnittstelle des Programmoduls nach außen fest. Die Zugriffsdeskriptoren im „privaten“ Teil können nur innerhalb des Moduls angesprochen werden. Diese Unterteilung, die vollständig dem Package-Konzept von ADA entspricht, erlaubt die saubere Trennung der Schnittstellendefinition eines Moduls von dessen Implementierung, ist also ein Mittel, um unnötige Informationen nach außen zu verbergen.

Contextobjekte definieren die dynamische Zugriffsumgebung eines Prozeduraufrufs. Das Contextobjekt wird beim Aufruf der Prozedur kreiert und bei der Rückkehr wieder zerstört, unterstützt damit also auf ideale Weise rekursive und Reentrant-Prozeduren. Der Context enthält auch ein Datenobjekt für die lokalen Daten und einen Operandenstock.

Jeder Context ist mit einer Zugriffsliste verbunden. Befehle innerhalb des Contexts können nur Objekte adressieren, die über diese Zugriffsleiste erreichbar sind. Die damit festgelegte Zugriffsumgebung ist vollständig mit dem Gültigkeitsbereich von Namen in verschiedenen Programmiersprachen in Einklang zu bringen: In ADA wird die Erreichbarkeit von Variablen und Unterprogrammen durch die Package-Struktur bestimmt, in rein blockstrukturierten Sprachen wie ALGOL durch die umfassenden Blöcke und in prozeduralen Sprachen wie FORTRAN ist im Prinzip aus jeder Anweisung der gesamte Programmbereich

ansprechbar. Damit kann ein Übersetzer für jede Programmiersprache die entsprechenden Zugriffslisten aufbauen.

3.4 Betriebssystem in Silizium

Die wohl markanteste Eigenschaft der 432-Architektur ist, daß Funktionen, die sonst im Betriebssystemkern zu finden sind, hier direkt in die Hardware integriert wurden. Grundlage dafür bilden die folgenden Systemobjekte, die den wichtigsten Betriebsmitteln und Systemdiensten entsprechen:

- Prozessorobjekt:
  Für jeden physikalischen Prozessor gibt es eine Datenstruktur im Speicher, die ihn genau beschreibt und ihm Informationen über seine Systemumgebung gibt.
- Prozeßobjekt:
  Jeder Prozeß (Task) im System ist durch solch ein Objekt beschrieben. Ein Prozeß ist eine Programmeinheit, die parall zu anderen Prozessen ausgeführt werden kann. Ein Prozeßobjekt enthält unter anderem Information über die Priorität und die Zeitscheibe des Prozesses.
- Portobjekt:
  Portobjekte sind Datenstrukturen, die die Funktion einer Warteschlange erfüllen. Sie können somit bei der Interprozeßkommunikation als Nachrichtenpuffer zwischen einzelnen Prozessen dienen, aber auch im Rahmen der Rechnerkernvergabe die lauffähigen Prozesse aufreihen. Die zweite Funktion wird dadurch erreicht, daß die Zuteilung eines Prozessors als das Senden einer Nachricht (des Prozeßobjekts) an einen physikalischen Prozeß (den Prozessor) interpretiert wird. In Portobjekten können die einzelnen Elemente in FIFO-Ordnung, aber auch entsprechend ihrer Priorität und/oder Zeitgrenze eingereiht werden.
- Storage-Resource-Objekt (SRO):
  Ein SRO ist ein Objekt, das die dynamische Freispeicherverwaltung im System unterstützt. Es beschreibt einen oder mehrere Teile des Arbeitsspeichers, aus dem Programme durch den Befehl „CREATE SEGMENT" neue Objekte generieren können. Die gesamte Suche innerhalb des SROs nach einem zusammenhängenden Stück ausreichender Größe, alle nötigen Verkettungen und die Übergabe des Zugriffdeskriptors für das neue Objekt werden von der Hardware vorgenommen. Zur Implementierung unterschiedlicher Zuteilungsstrategien können mehrere solcher Storage-Resource-Objekte im System existieren.

Ähnliche Datenstrukturen existieren sicher in fast allen Rechnern; der wesentliche Unterschied zum iAPX 432 besteht allerdings darin, daß die Hardware die einzelnen Objekte erkennt und Befehle sowie unsichtbare Funktionen darauf definiert sind. Die Zuteilung eines Prozessors an einen Prozeß (= *Dispatching*) sowie das Einsortieren eines Prozesses in die Warteschlange (*Dispatching Port*) der lauffähigen Programme (= *Scheduling*) werden in der 432-Architektur automatisch von den einzelnen Prozessoren ausgeführt, ohne daß auch nur ein einziger Befehl dafür erscheint. Dadurch ist auch das softwaretransparente Multiprozessing möglich; jeder freie Prozessor nimmt sich nämlich selbsttätig aus der Warteschlange der lauffähigen Prozesse den in der Reihenfolge ersten

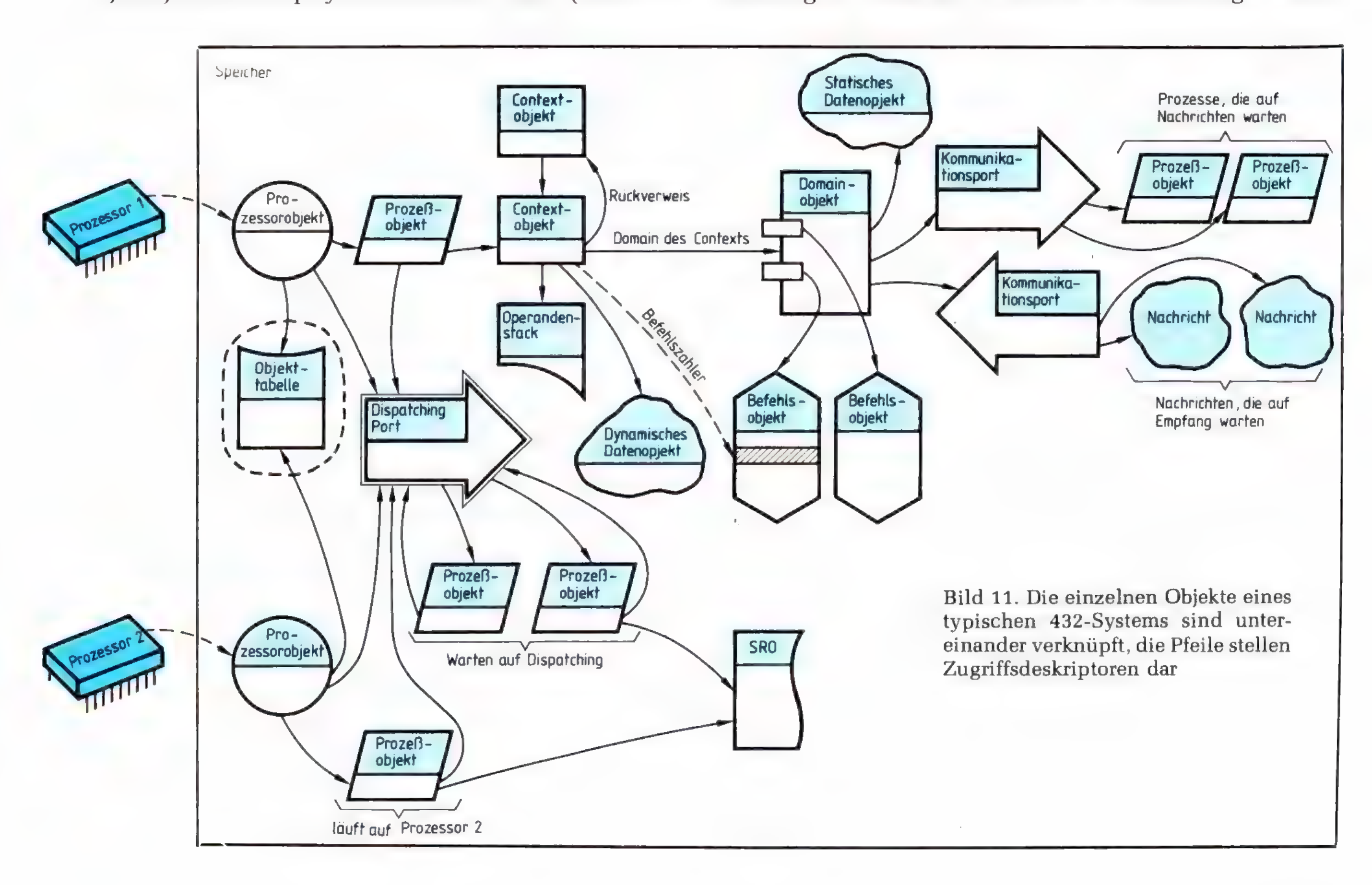


Bild 11. Die einzelnen Objekte eines typischen 432-Systems sind untereinander verknüpft, die Pfeile stellen Zugriffsdeskriptoren dar

```
PASCAL                          ADA
type
  Weekday = (Mon, Tue,          type Weekday is (Mon, Tue,
      Wed, Thu, Fri, Sat, Sun);      Wed, Thu, Fri, Sat, Sun);
  Date = record                 type Date is
                                  record
    Day: 1..31;                     Day: integer range 1..31;
    Month: 1..12;                   Month: integer range 1..12;
Year: 0..4000;                      Year: integer range 0..4000;
    DayOfWk: Weekday;               DayOfWk: Weekday;
    end;                        end record;
  DatePointer = ^ Date;         type DatePointer is access Date;
Var
  D: Date;                      D: Date;
  D2: array (1..20) of Date;    D2: array (1..20) of Date;
```

◀ Bild 12.
Viele Sprachelemente von ADA gehen auf PASCAL zurück, speziell Vereinbarungen und Ausdrücke sind sehr ähnlich

Prozeß heraus und beginnt mit seiner Ausführung. Er braucht dazu überhaupt nicht zu wissen, wie viele andere Prozessoren im System existieren. Nach Ablauf der Zeitscheibe eines Prozesses oder beim Auftreten einer Wartebedingung sortiert der Prozessor das Prozeßobjekt in die entsprechende Warteschlange ein und ist wieder frei.

Das Portmodell dient auch zur Übertragung von Nachrichten zwischen asynchron zueinander ablaufenden Prozessen. Ein Port wirkt als Zwischenpuffer für Nachrichten, deren Empfänger noch nicht bereit ist, die Nachricht zu übernehmen; er ist eventuell noch gar nicht gestartet. Ähnlich wird auch ein Empfängerprozeß, der aus einem im Augenblick leeren Port empfangen möchte, von seinem ausführenden Prozessor angehalten und an den Port „angebunden"; der Prozessor ist anschließend frei und kann wieder mit dem Dispatching beginnen. Sobald zu einem späteren Zeitpunkt ein Prozessor bei der Ausführung eines Prozesses durch den Befehl „SEND MESSAGE" eine Nachricht an diesen Port schickt, legt der Prozessor den Verweis auf die Nachricht direkt im Prozeßobjekt des wartenden Prozesses ab und ordnet es auch automatisch in die Warteschlange der lauffähigen Prozesse ein.

Im *Bild 11* wird ein Überblick über die wichtigsten Objekte in einem 432-System und ihr Zusammenhang untereinander gegeben. Für viele Objekte gibt es Analogien zu konventionellen Rechnern; der gravierende Unterschied besteht darin, daß der iAPX 432 alle diese Objekte mit Hilfe von Hardware erkennt und die Operationen darauf einzelne Befehle sind oder sogar automatisch ablaufen, womit die Architektur des iAPX 432 speziell für diese Funktionen um ein Vielfaches schneller ist als jeder andere Rechner.



Dipl.-Inf. Johann Geyer studierte an der TU in München Informatik mit dem Schwerpunkt Betriebssysteme für Mikroprozessoren. Nach seinem Abschluß 1977 arbeitete er bei der Siemens AG in München, wo er im Bereich der Basis-Informationssysteme Terminals, Hard- und Software entwickelte. Seit April 1980 ist er bei der Firma Intel als Applikationsingenieur für die 16- und 32-Bit-Mikrocomputerfamilien zuständig.
Hobbys: Bergwandern, Tauchen
Dienstttel.: (0 89) 53 89-1.


```
PASCAL      ADA
            package Processes is
Nicht         type process is private;
verfügbar     type priority is integer range 1..50;
              function Create(p:priority) return process;
              procedure Destroy(p:process);
            end Processes;
```

Bild 13. Das Packagekonzept in ADA unterstützt die Modularisierung von großen Programmpaketen

4 Programmierumgebung

Zur Erstellung von Software für den iAPX 432 dient die Programmiersprache ADA. Diese Sprache ist nach einem fünfjährigen Auswahlprozeß aus einem Projekt des amerikanischen Verteidigungsministeriums hervorgegangen und im August 1980 endgültig verabschiedet worden [7]. Neben den guten leistungsfähigen Eigenschaften der Sprache selbst wird vor allem die Entscheidung des amerikanischen Verteidigungsministeriums, ab 1983 neue Software nur noch in ADA zu akzeptieren, dazu führen, daß sich ADA sehr schnell und in breiter Front durchsetzen wird.

ADA wurde speziell für die Erstellung großer Softwarepakete konzipiert und erinnert in vielen Teilen sehr stark an PASCAL (*Bild 12*), bringt jedoch einige wichtige neue Eigenschaften:

- Multitasking und Prozeßkommunikation werden unterstützt.
- Das Packagekonzept erlaubt die saubere Modularisierung von großen Programmpaketen. Durch die Trennung der Schnittstellendefinition eines Pakkage von seiner eigentlichen Implementierung wird das Projektmanagement großer Softwarepakete erleichtert (*Bild 13*).

Obwohl ADA und das Konzept des iAPX 432 vollkommen getrennt voneinander entstanden sind, kann man ihre Strukturen genau aufeinander abbilden:

- Ein ADA-Objekt entspricht einem 432-Objekt.
- Ein ADA-Access entspricht beim iAPX 432 einer Objektreferenz.
- Ein ADA-Package entspricht einer 432-Domain.
- Die Aktivierung eines Unterprogramms in ADA entspricht dem Context beim iAPX 432.

Aus unterschiedlichen Richtungen sind also zwei unabhängige Gruppen von Entwicklern zum gleichen Ergebnis gekommen. Auch dies unterstreicht die Überlegenheit der Architektur des iAPX 432.

Literatur

1 *Lattin, W., Mathiasen, T., Grovender, S.*: 64-pin QUIP keeps microprocessor chips cool and accessible. Electronics, 1981.

2 *Parnas, D. L.*: On the criteria to be used in decomposing systems into modules. Comm. ACM 15, 12 (Dec. 1972).

3 *Hoare, C. A. R.*: Notes on Data Structuring in „Structued Programming", *O.-J. Dahl, E.W. Dijkstra, C. A. R. Hoare*, Academic Press, New York 1972.

4 *Linden, T. A.*: Operating System Structures to Support Security and Reliable Software. Computing Surveys, Vol. 8, No. 4, Dec. 1976.

5 *Fabary, R.S.*: Capability-based addressing. Comm. ACM 17, 7 (July 74).

6 8086 Family User's Manual. Numerics Supplement. Intel, July 80.

7 *Ichbiah, J.*: ADA Reference Manual. SIGPLAN Notices, July 80.

Dipl.-Math. Reiner Mauthe

Mehrprozessor-Unterstützung in Echtzeit-Betriebssystemen

Man unterscheidet im wesentlichen zwei Arten der Mehr-Prozessor-Technik: transparentes und nichttransparentes Multiprocessing. Transparent bedeutet, daß sich die Prozessoren in einem System die gesamten Aufgaben teilen, kein Prozessor wird bevorzugt. Vorteil dieses Verfahrens ist die Möglichkeit, den Durchsatz des Systems zu steigern, indem man weitere Prozessoren hinzufügt. Der Ausfall eines Prozessors bedeutet nur eine Leistungsminderung, aber keine Funktionsstörung. Es gibt nur ein Betriebssystem für die Gesamtkonfiguration, das die Prozessoren als gemeinsame Mittel verwaltet. Der Verwaltungsaufwand im Betriebssystem wird wesentlich erhöht, wenn solche Funktionen nicht direkt in der Hardware implementiert sind. Ein Beispiel eines transparenten Multiprozessor-Systems mit Scheduling-Funktionen, die in der Hardware direkt implementiert sind, ist der iAPX 432 [1]. Nichttransparentes Multiprocessing bedeutet, daß jeder Prozessor im System eine spezielle Aufgabe übernimmt. Jeder Prozessor hat sein eigenes Betriebssystem. Im Fehlerfall ist er damit nicht in der Lage, die Rolle eines anderen Prozessors zu übernehmen. MMX800 [2] bietet die Möglichkeit der Kommunikation zwischen den Betriebssystemen RMX/80 [4], RMX/88 [5] und RMX/86 [3]. Es ist damit möglich, sowohl ein 16-Bit-Mehrprozessorsystem zu realisieren als auch ein gemischtes 8- und 16-Bit-System aufzubauen.

Leistungssteigerung durch mehrere Prozessoren

Es gibt zwei Möglichkeiten, die Leistungsfähigkeit eines Systems zu erhöhen:

1. Den existierenden Prozessor durch einen leistungsfähigeren ersetzen;
2. weitere Prozessoren hinzufügen.

Die erste Alternative bedeutet normalerweise das Umsteigen auf einen größeren Rechner. Dies ist meistens sehr kostspielig, da sowohl Teile der Hardware geändert werden müssen, als auch die Software, um kompatibel zum neuen Rechner zu sein.

Dieses Problem könnte vermieden werden, wenn von Beginn an ein größerer Rechner vorgesehen wird, um für zukünftige Erweiterungen gerüstet zu sein. Aber ein solches Konzept ist meistens zu kostspielig und unter der Randbedingung möglichst schnell ein Produkt zu entwickeln, nicht durchsetzbar. Die zweite Alternative, weitere Prozessoren hinzufügen, ist für den Benutzer nur dann sinnvoll, wenn in seiner Anwendung mehrere Tasks existieren. die gleichzeitig mit dem Prozessor arbeiten könnten. In einer RMX-Umgebung sind dies alle Tasks, die sich im laufenden Zustand oder sich in der „Ready"-Warteschlange befinden. Unter diesen Umständen ist es möglich, diese Tasks auf mehrere Prozessoren zu verteilen und gleichzeitig ablaufen zu lassen.

Die meisten Echtzeit-Systeme enthalten Tasks, die gleichzeitig ablaufen könnten. Diese bearbeiten asynchrone Ereignisse der Außenwelt. Deshalb haben die Tasks nicht notwendigerweise eine strikte Reihenfolge, in der sie abgearbeitet werden müssen. In solchen Fällen eignet sich der Einsatz mehrerer Prozessoren in einem Echtzeit-System.

Das Übertragen von Software von einem Ein- auf ein Mehrprozessorsystem bringt natürlich neue Probleme mit sich. Obwohl Echtzeit-Tasks i. a. im Verhältnis zueinander asynchron ablaufen, ist in vielen Fällen eine Kommunikation zwischen diesen Tasks durchzuführen. Deswegen muß auch die Möglichkeit einer Inter-Task-Kommunikation in einer Mehrprozessorumgebung vorhanden sein, was das ursprüngliche Softwarekonzept nicht sehr stark beeinflussen sollte. In einer Multibus-Umgebung wird ein Mehrprozessorsystem durch die Verwendung mehrerer Einplatinen-Computer realisiert. Der Zugriff auf den Multibus durch diese Boards wird durch eine Hardware-Entscheidungslogik geregelt. Da jeder dieser Computer sein eigenes Betriebssystem besitzen kann, ist eine Kommunikation zwischen diesen Boards in der Regel nicht vorhanden. Die folgenden Abschnitte behandeln die Software MMX800 und den Zusammenhang zu den RMX-Betriebssystemen. Damit wird dem Benutzer die Möglichkeit gegeben, Inter-Task-Kommunikation zwischen Einplatinen-Computern zu

betreiben, um damit die Vorteile einer Mehrprozessorumgebung auszunutzen.

Software MMX800

Die „Multibus Message Exchange Software“ MMX800 [2] ist eine logische Erweiterung der Inter-Task-Kommunikations-Mechanismen des Betriebssystems RMX/86. Die sendende Task schickt eine Nachricht an eine „Mailbox“ an der die empfangende Task wartet. Die Implementierung des Nachrichtenaustauschs ist natürlich unterschiedlich zu RMX/86, da es die Zusammenarbeit zweier Betriebssysteme erfordert. Die Systemaufrufe für MMX800 zeigen dies:

ACTIVATE PORT

stellt für die empfangende Task die Verbindung zwischen der Mailbox auf der Platine und dem global bekannten Systemportnamen her. An dieser Mailbox können dann die Nachrichten empfangen werden, die von Tasks auf anderen Boards geschickt werden.

FIND PORT

liefert der sendenden Task eine Identifikation (Token) zurück, die es mit dem TRANSFER-Aufruf erlaubt, eine Nachricht an eine Mailbox zu schicken. Diese Mailbox stellt wiederum die Verbindung zum globalen Systemport dar.

TANSFER MESSAGE

leitet einen MMX800-Nachrichtentransfer ein und schickt eine Nachricht an den angegebenen Systemport.

LOSE PORT

hebt die Verbindung, die durch „FIND PORT“ zustande gekommen ist, wieder auf.

DEACTIVATE PORT

ist das entsprechende Gegenteil zu „ACTIVATE PORT“.

Um eine Nachricht an eine Task auf einem anderen Computer zu schicken, wird zuerst ein Aufruf „FIND PORT“ mit dem gewünschten Systemportnamen abgesetzt. Die Task setzt dann einen Aufruf „TRANSFER MESSAGE“ für jede zu sendende Nachricht ab. Ein Parameter dieses Aufrufs ist die vom Betriebssystem zurückgegebene Identifikation für den Systemport, an den diese Nachricht geschickt wird. Um eine Nachricht zu empfangen, setzt die Task zuerst einen Aufruf „ACTIVATE PORT“ ab. Die Task wartet danach an der Mailbox, deren Adresse durch den Aufruf vom Betriebssystem zurückgegeben wird. Weitere Nachrichten können an der gleichen Mailbox durch zusätzliche Aufrufe empfangen werden.

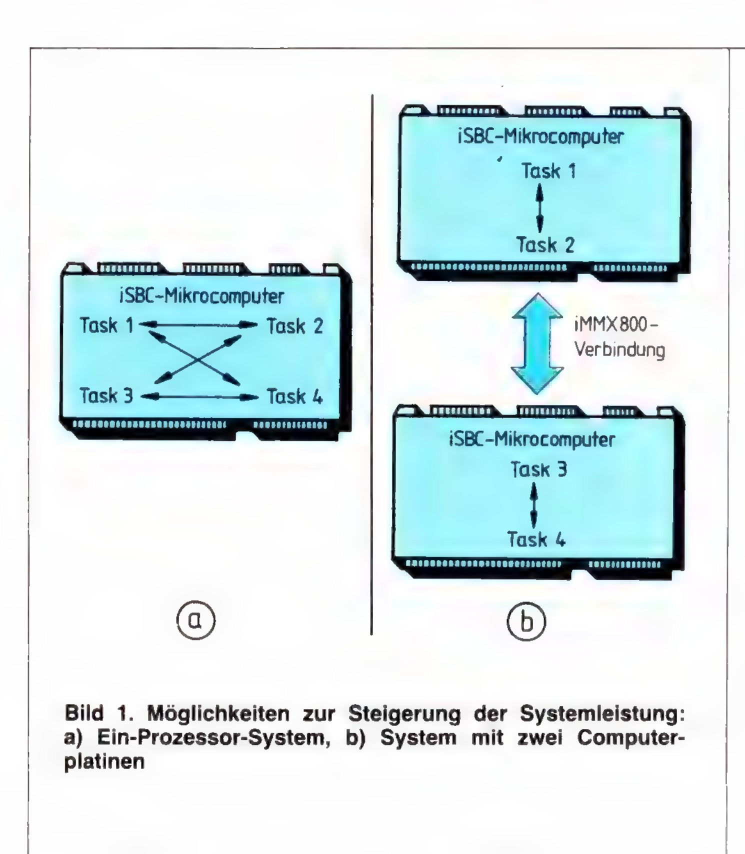


Bild 1. Möglichkeiten zur Steigerung der Systemleistung: a) Ein-Prozessor-System, b) System mit zwei Computerplatinen

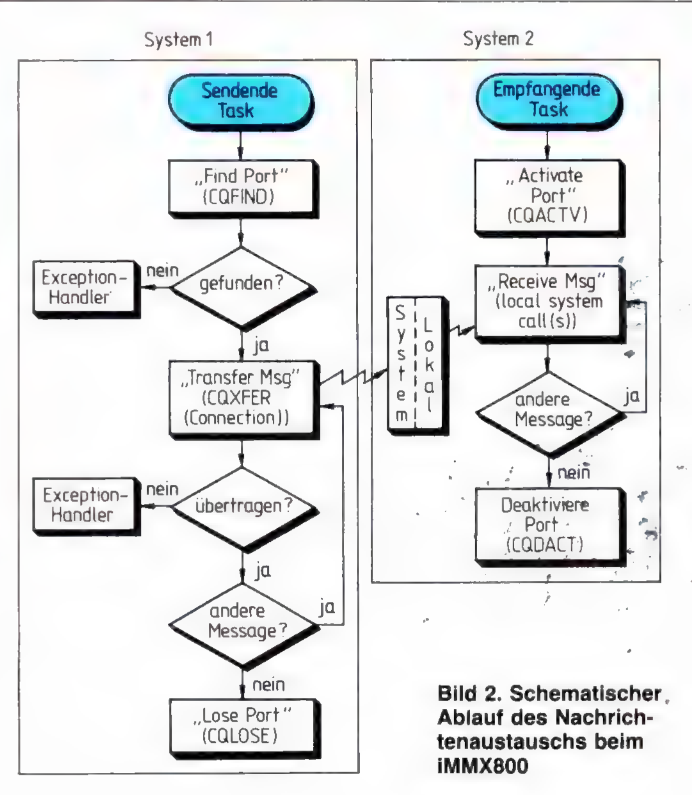


Bild 2. Schematischer Ablauf des Nachrichtenaustauschs beim iMMX800

„LOSE PORT“ und „DEACTIVATE PORT“ sind nicht für den Meldungsaustausch notwendig. Sie geben nur die Betriebsmittel wieder an das Betriebssystem zurück. Ein schematischer Ablauf ist in *Bild 2* dargestellt.

Beispiel eines Mehrprozessorsystems

Bild 1a stellt eine Konfiguration für eine Echtzeit-Anwendung mit vier Tasks dar, die auf einem Einplatinen-Computer ablaufen. Es wird angenommen, daß diese Tasks zusammen den Prozessor zu 100 Prozent auslasten, d. h., keine CPU-Zeit vergeudet wird. Wenn man annimmt, daß z. B. Task 1 und 3 sowie Task 2 und 4 häufig gleichzeitig im „Ready-“ bzw. „Runnig-“ Zustand sind, heißt das, daß diese Kombinationen logisch gleichzeitig ablaufen könnten. Um hier nun einen höheren Durchsatz zu erreichen, wählt man eine Konfiguration, wie sie *Bild 1b* zeigt. Task 1 und 3 sowie Task 2 und 4 sind nun auf verschiedenen Rechnern, was ihnen auch physikalisch die gleichzeitige Ausführung erlaubt. Da Task 2 und 3 bzw. Task 1 und 4 auch Inter-Task-Kommunikation benötigen, wird MMX800 verwendet. Der tatsächliche Gewinn an Durchsatz ist nicht der Faktor 2, weil hier zwei Prozessoren beteiligt sind. Er hängt von zusätzlichen Randbedingungen ab, wie z. B. die Prozessorzeit, die jede einzelne Task für sich in Anspruch nimmt.

Modularität eines Systems

Eine Tendenz eines Mehrprozessorsystems ist die Spezialisierung der einzelnen Subsysteme für eine bestimmte Aufgabe. Es kann eine Peripherieeinheit eines Einplatinen-Computers als Datenbasis für das gesamte Multibussystem benutzt werden. Ebenfalls kann eine Numerikeinheit (hier SBC 337 mit 8087) Floating-Point-Funktionen für das Gesamtsystem zur Verfügung stellen. Dieses Konzept eines Mehrprozessorsystems kann durch den Einsatz des Ethernet-Kommunikationsboards SBC 550 [6] noch erweitert werden (*Bild 3*). Alle MMX800-Verbindungen benutzen auf dem Multibus das gleiche Hardwareprotokoll. Dieses Protokoll MIP (Multibus Interprozessor Protocol) ist in [2] beschrieben.

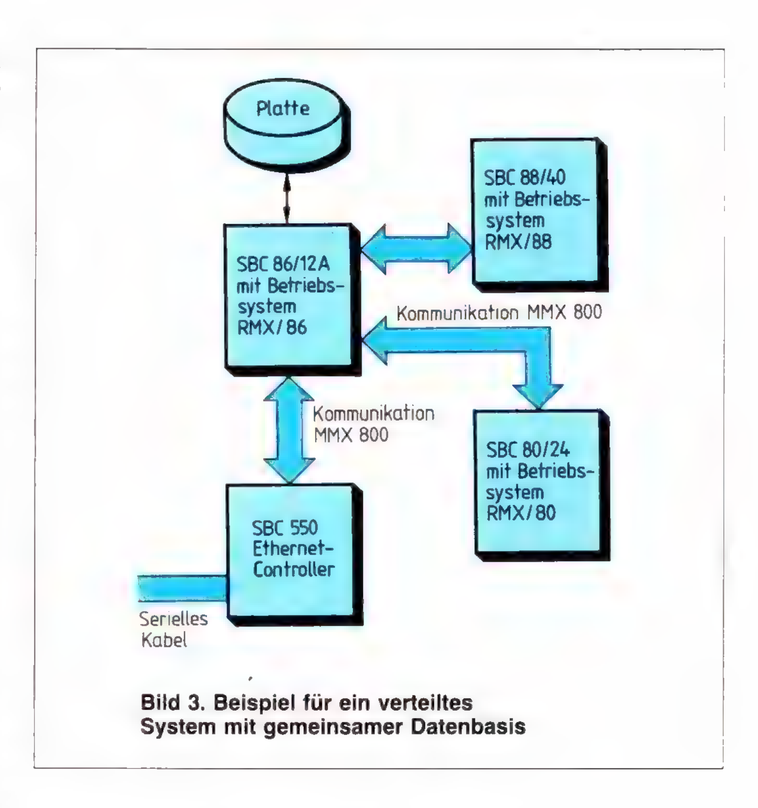


Bild 3. Beispiel für ein verteiltes System mit gemeinsamer Datenbasis

Zusammenfassung

Die RMX-Betriebssysteme zusammen mit der Erweiterung MMX800 erlauben einen modularen Aufbau einer Echtzeit-Anwendung. Ein Multibus-System kann ohne prinzipielles Redesign des Gesamtsystems schrittweise aufgerüstet werden. Das System kann man in funktionelle Untersysteme aufteilen, die ihre speziellen Dienste dem Gesamtsystem zur Verfügung stellen. Auf diese Weise sind auch zukünftige Entwicklungen einfach in das gesamte System zu integrieren.



Dipl.-Math. Reiner Mauthe wurde in Konstanz geboren und war nach Abschluß seines Studiums bei Siemens in München beschäftigt. Dort war er für die Entwicklung von Software für Vermittlungssysteme zuständig. Seit 1979 ist er bei Intel in München als Applikations-Ingenieur für Entwicklungssysteme, Compiler und Betriebssysteme verantwortlich.
Hobbys: Reisen, Skilanglauf

Literatur

[1] Geyer J.: 32-Bit-Microcomputer besitzt neuartige Architektur. ELEKTRONIK 81, H. 5, S. 59...66.
[2] MMX800 Software Reference Manual and User's Guide. Intel Corporation, 143808-002.
[3] RMX/86 Nucleus Reference Manual. Intel Corporation, 9803122-003.
[4] RMX/80 User's Guide. Intel Corporation, 9800522-06.
[5] RMX/88 Reference Manual. Intel Corporation, 143252-002.
[6] Ethernet Communications Controller Programmers Reference Manual. Intel Corporation, 121769-001.

Dipl.-Ing. Jens-Peter Gast

IC steuert zweiseitigen Speicher

Leistungsfähige Mikrocomputer ermöglichen die Projektierung von Prozeßsteuerungen und -regelungen mit Eigenschaften, die den bisherigen Lösungen in konventioneller Hybridtechnik überlegen sind. Dies gilt vor allem auch für Funktionserweiterungen zur Erleichterung von Inbetriebnahme, Änderungen, Wartung sowie für Maßnahmen zur Erhöhung der Verfügbarkeit. Zur Abdeckung der verschiedenen Anforderungen eines Aufgabenspektrums ist ein hoher Grad von Modularität bei Hard- und Software notwendig. Dazu gehört auch die Anwendungsmöglichkeit eines Hardwaremenüs in verschiedenen Multiprozessor-Konfigurationen [1].

Eine Art der Rechnerkopplung, die Speicherkopplung, ist im Rahmen einer vom Bundesminister für Forschung und Technologie (BMFT) geförderten Arbeit – Entwicklung einer Datenübertragung auf Bahnfahrzeugen über Lichtleiter – zunächst für dieses und ähnliche Aufgabenspektren [2, 3] entwickelt worden. Das Ergebnis, ein einziger Integrierter Schaltkreis zur Steuerung der Speicher, aller Daten- und Adreßbuspuffer sowie der beteiligten Mikroprozessoren, ist universell, auch für verschiedene Prozessortypen, anwendbar. Die Steuerung benötigt keine Software.

1 Aufgaben der Steuerung eines zweiseitigen Speichers

Der Aufbau eines zweiseitigen Speichers, eines von zwei Mikrocomputern erreichbaren Schreib- und Lesespeichers, wird in *Bild 1* dargestellt. Die beteiligten Prozessoren müssen nicht taktsynchron und frequenzgleich arbeiten.

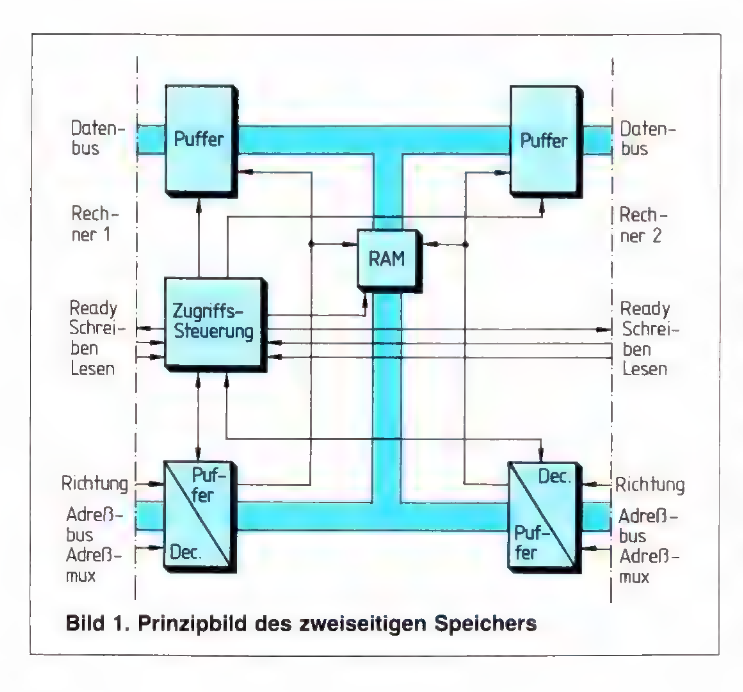


Bild 1. Prinzipbild des zweiseitigen Speichers

Die Aufgaben der Zugriffssteuerung ergeben sich aus der Anordnung:
- Erkennung der Zugriffswünsche;
- Vergabe der Priorität an einen Prozessor;
- Sperren des Konkurrenten, jedoch nur dann, wenn dieser während eines laufenden Zugriffs des priorisierten Rechners ebenfalls zugreifen will;
- Steuerung der Datenbuspuffer-Schnittstellen;
- Steuerung der Adreßbuspuffer-Schnittstellen;
- Steuerung der Speicherblöcke.

Daraus ergibt sich die innere Struktur des Steuerbausteins, die in *Bild 2* dargestellt ist.

2 Funktionsbeschreibung

Bei zwei im allgemeinen nicht taktsynchron und frequenzgleich arbeitenden Mikrocomputern, die auf einen gemeinsamen Speicherbereich konfliktfrei zugreifen sollen, hat man bei rein hardwareseitiger Steuerung der Zugriffe nur eine Möglichkeit, bei gleichzeitigem Lesen und Schreiben von beiden Rechnern die Priorität zu vergeben: Der zeitlich zuerst zugreifende Prozessor muß seinen Zugriff ungestört zu Ende durchführen können.

Kriterium für die Zugriffswünsche sind die aus den Adreßleitungen decodierten SELECT-Signale der Bereiche des gemeinsamen Speichers. Sie sind auch ein Maß für die Dauer der Zugriffe.

Der Funktionsblock „Priorität" (Bild 2) ordnet jedem der beiden Prozessoren, der zeitlich zuerst zugreift, den internen Status „FAC" (Erster) zu. Das bedeutet für

diesen Prozessor einen ungestörten Ablauf des Zugriffes mit entsprechender Steuerung der zugehörigen Buspuffer und Speicherselect-Signale. Der Konkurrent bleibt ebenfalls ungestört, solange er nicht auf einen gemeinsamen Speicherbereich zugreifen will. Wird jedoch aus dessen Adreßleitungen eine dafür gültige Adresse decodiert, also ein Zugriffswunsch erkannt, erhält dieser Prozessor sofort den Status „NOT READY" mit der Folge, daß die Steuerleitung „READY" auf logisch „0" geht. Er kann während eines laufenden Zugriffes des Prozessors mit dem Status „FAC" nicht mehr ungestört zugreifen. Daher werden auch die Steuersignale für die zugehörigen Buspuffer und den angesprochenen Speicherblock während „NOT READY" unterdrückt. „NOT READY" bewirkt bei dem betroffenen Prozessor, nach taktabhängiger Erkennung, einen Übergang in den „WAIT STATE"; nach außen bedeutet dies sozusagen ein „Einfrieren" aller Zustände der Bus- und Steuerleitungen des unterbrochenen Prozessors. Dies bedeutet aber auch, daß dessen Zugriffswunsch, also die gültige Adresse im gemeinsamen Speicherbereich, weiter ansteht.

Beendet der Prozessor, der den Status „FAC" erhalten hatte, nun seinerseits den Zugriff, indem dessen Adreßbus z. B. die Adresse des nächsten zu decodierenden Befehls im Programmspeicher annimmt, erhält nunmehr sofort der bisher wartende Prozessor den Status „FAC". Damit wird er „READY" und verläßt, nach wiederum taktabhängiger Auswertung dieses Steuersignales, den „WAIT STATE" und führt den unterbrochenen Zugriff zu Ende, da nun auch die Unterdrückung der Steuerung der Puffer und des Speicherselect-Signales intern aufgehoben ist.

Der Sonderfall des absolut gleichzeitigen Zugriffes, mit einem vorhergehenden Bereich des nahezu gleichzeitigen Zugriffes, wird intern auf zwei verschiedenen Wegen gelöst:

- Im Grenzbereich des nahezu gleichzeitigen Zugriffes kann durch externe Verbindung jeweils zweier von drei Anschlüssen der eine oder der andere Prozessor bevorzugt werden. Gleichzeitig kann durch entsprechende Beschaltung der beiden Steuereingänge durch externe Logik der eine oder der andere Prozessor nach dem ersten Zugriff in einen dauernden „WAIT STATE" versetzt werden. Dieser „WAIT STATE" dauert dann solange, bis der Steuereingang wieder freigegeben wird, eine Option, die bei „Direktem Speicherzugriff" (DMA) in den Koppelspeicher benötigt wird.
- Bei zufällig absolut gleichzeitigem Zugriff reagiert die Logik der Zugriffssteuerung derart, daß derjenige Priorität für ungestörten Zugriff behält, der auch beim letzten, zeitlich davorliegenden Konflikt den Status „FAC" zuletzt erhalten hatte.

Das Anschlußbild des Integrierten Schaltkreises zeigt *Bild 3*. Die Schaltung ist in CMOS-Metal-Gate-Technik ausgeführt und in einem 28poligen Keramikgehäuse untergebracht. Alle Signale sind TTL-kompatibel, der Schaltkreis ist für den erweiterten Temperaturbereich spezifiziert.

Die Signalgruppen haben folgende Bedeutung:

$\overline{\text{xCSRAMAK}}$, $\overline{\text{......BK}}$, $\overline{\text{......CK}}$, $\overline{\text{......DK}}$ — Low-aktive Signale aus einem Adreßdecoder, z. B. einem schnellen bipolaren PROM, für maximal vier Adreßblöcke;

$\overline{\text{xREADY}}$ — Open-Collector-Ausgänge für die entsprechenden Eingänge der Prozessoren;

$\overline{\text{xADRBUSY}}$ — Low-aktive Signale zur Steuerung der Adreßbuspuffer, z. B. 'LS244;

$\overline{\text{xDATABUSY}}$ — Low-aktive Signale zur Steuerung der Datenbuspuffer, z. B. 'LS245;

$\overline{\text{xMWR}}$ — Low-aktive Eingangssignale, Bussteuersignale „Speicherschreiben" der Prozessoren;

$\overline{\text{xMRD}}$ — Low-aktive Eingangssignale, Bussteuersignale „Speicher lesen" der Prozessoren;

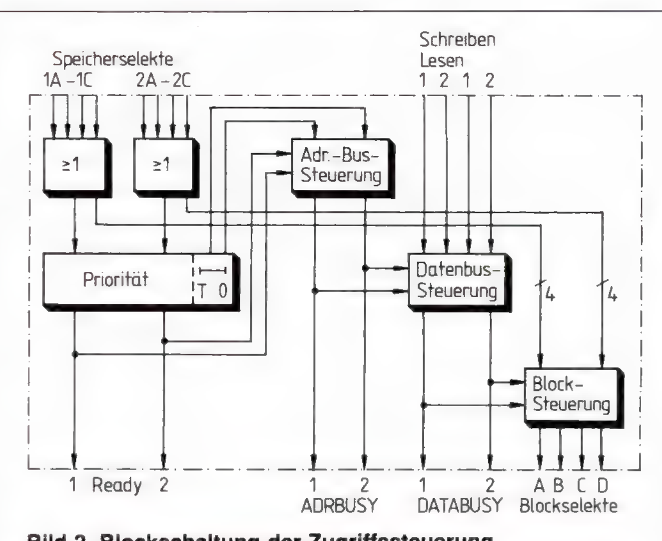


Bild 2. Blockschaltung der Zugriffssteuerung

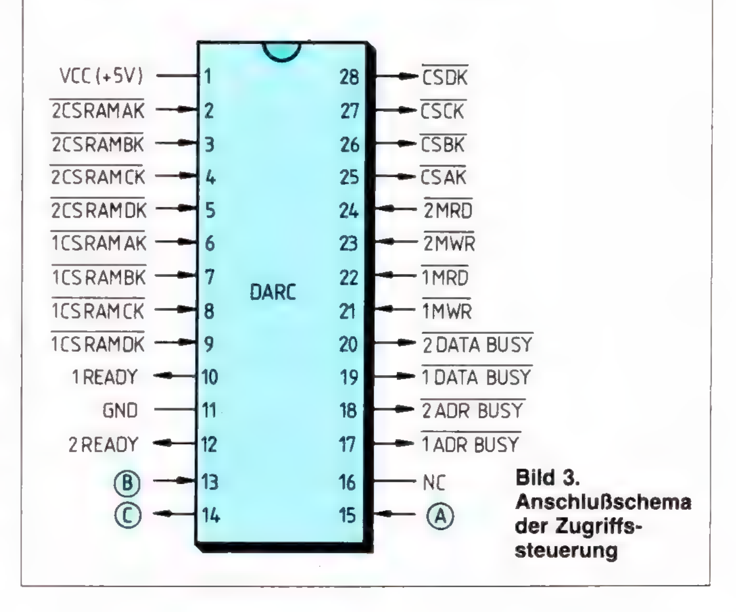


Bild 3. Anschlußschema der Zugriffssteuerung

$\overline{\text{CSAK}}$ $\overline{\text{..BK}}$ $\overline{\text{..CK}}$ $\overline{\text{..DK}}$	Speicherselect-Signale, Low-aktive, an maximal vier Adreßblöcke
C	Grenzbereichsausgang „Beide READY Low“
A, B	Prozessor 1 (A) oder 2 (B) „NOT READY“ ab nächstem Zugriff für die Dauer des Low-aktiven Eingangssignales; zugleich zusätzliche „FAC“-Setzeingänge im Grenzbereich.

Einen Prototyp des Integrierten Schaltkreises „DARC“ zeigt *Bild 4*.

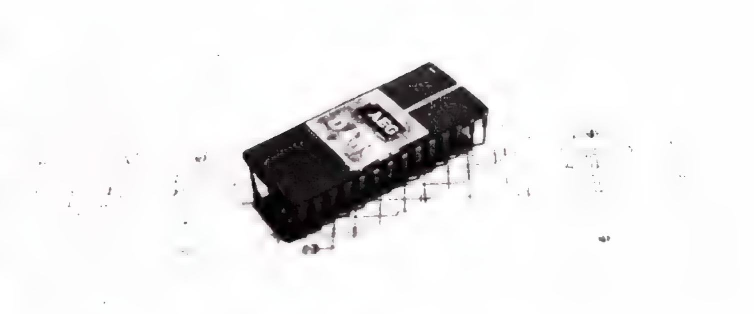


Bild 4. Zugriffssteuerung „DARC“ als Integrierter Schaltkreis (Foto: AEG/J.-P. Gast)

3 Anwendungen

Es ist häufig wirtschaftlicher, hohen Datendurchsatz nicht mit einem schnellen und daher überproportional teuren, sondern mit mehreren 8- oder 16-Bit-Mikrocomputern zu realisieren [1]. Dies gilt vor allem dann, wenn auch die zu steuernden und regelnden Prozesse in klar trennbare Teilbereiche aufgegliedert werden können, die so dann auch auf die einzelnen Bereichsrechner hard- und softwareseitig verteilt werden können. Hier bietet sich die Speicherkopplung als Bindeglied deshalb besonders an, weil sie im Gegensatz zu allen anderen bekannten Kopplungsmöglichkeiten keinerlei Software für diese Funktion benötigt. Die Kopplung der Teilprozesse erfolgt allein über vereinbarte Datenspeicheradressen in beiden Richtungen.

Bezogen auf die Zahl der Speicherzugriffe ist darüber hinaus auch die Zahl der Konflikte klein, so daß der Verlust an Operationszeit durch „WAIT STATES“, und dies ohnehin dann nur für die restlichen Zugriffszeiten der jeweiligen Konkurrenten, praktisch unbedeutend ist.

Je nach Anordnung des Gesamtsystems, lineare oder ringförmige Anordnung gleichberechtigter Mikrocomputer oder sternförmige Anordnung am Bussystem eines zentralen Rechners, ergeben sich verschiedene vom Aufgabenspektrum abhängige Möglichkeiten.

Die Aufteilung von Teilbereichen eines Prozesses auf speichergekoppelte Bereichsrechner ergibt noch einen weiteren Vorteil, der vor allem dort greift, wo Betriebsmittel und, besonders wichtig, Menschen vor den Folgen von Fehlfunktionen unbedingt geschützt werden müssen (z. B. bei Bahnanwendungen).

Einige wenige zusätzliche Schnittstellen und Zusatzsoftware zur gegenseitigen Überwachung aller Teilsysteme über die Speicherschnittstellen gestatten bei Teilausfällen deren Erkennung und Durchführung von abgestuften Reaktionen – Störungsmeldungen, automatische Reduktion von Führungsgrößen, Sperrung aller Energiezufuhr zum Gesamtprozeß, Gefahrenbremse usw. – zur Sicherung von Menschenleben und Betriebsmitteln bzw. zur Erhöhung der Verfügbarkeit von Anlagen durch Betrieb mit automatisch eingeschränkten Eigenschaften bis zur Beseitigung der Teilausfälle.

Die diesem Bericht zugrunde liegenden Arbeiten wurden vom BMFT gefördert (Förderungskennzeichen TV 8040A). Die Verantwortung liegt jedoch allein beim Autor.



Dipl.-Ing. J.-P. Gast studierte in seiner Heimatstadt Berlin an der Technischen Universität Elektrotechnik. Anfang 1975 trat er in die AEG ein, wo er sich seitdem, abgesehen von einer kurzen Unterbrechung, mit der Entwicklung und Erprobung von Geräten zur Steuerung und Regelung elektrischer Bahntraktion beschäftigt. In den letzten Jahren drang die Mikroelektronik auch auf dieses Gebiet vor und erforderte entsprechende Entwicklungsarbeiten und Engagement.
Hobbys: Beschäftigung mit den beiden Töchtern, Fotografie einschl. Labor, Ersatz möglichst aller Handwerksberufe im Haushalt.

Literatur

[1] *Schmid, H.*: Multi-Mikroprozessor-Systeme. ELEKTRONIK 1982, H. 2, S. 87...95.

[2] *Gast, J.-P.*: Fahr-/Brems-Steuerung mit Mikrocomputern für Triebfahrzeuge – Erprobung in einem U-Bahn-Zug. DER STADTVERKEHR 26 (1981), H. 2, S. 59...61.

[3] *Gast, J.-P.*: Einsatz moderner Mikroelektronik auf Bahnfahrzeugen. ELEKTRISCHE BAHNEN 79 (1981), H. 12, S. 406...411.

Dipl.-Ing. Wilfried Heer

FIO-Baustein erleichtert Prozessorkopplung und Peripherieanschluß

Multiprozessorsysteme lassen sich durch Zugriff auf gemeinsame Speicher verkoppeln. Zu diesem Zweck kann man konventionelle Speichersysteme aufbauen und mit einer entsprechenden Steuerung ausstatten. Inzwischen sind aber die ersten integrierten Bausteine verfügbar, mit denen man die Kopplung in Mehrprozessor-Systemen realisieren kann. Weiteres Anwendungsgebiet solcher Bausteine ist der Anschluß langsamer Peripherie an Mikrocomputer. Im folgenden Beitrag werden Aufbau und Anwendungen des FIO-Bausteins Z8038 aus der 16-Bit-Mikroprozessor-Familie Z8000 beschrieben.

1 Mehrprozessor-Systeme

Der Leistungssteigerung bei Mikrocomputer-Bauelementen steht eine immer höhere Anforderung seitens der Anwender gegenüber. Bauelementeleistung ist jedoch aus physikalischen Gründen nicht unbegrenzt. Weitere Leistungssteigerung wird nur noch durch eine Verbesserung der Systemarchitektur möglich sein, z. B. durch Parallelverarbeitung, d. h. Mehrprozessor-Systeme.

Obwohl die physikalische Leistungsgrenze bei den heutigen Mikroprozessor-Bausteinen noch lange nicht erreicht ist, sind heute Mehrprozessor-Systeme bereits Stand der Technik. Diese Systeme können in zwei Gruppen klassifiziert werden; in
- festgekoppelte Mehrprozessor-Systeme und
- lose gekoppelte Mehrprozessor-Systeme.

Innerhalb dieser Gruppen könnte man noch weiter unterteilen, z. B. in Master-Slave-Systeme, homogene Systeme, die aus einer Anzahl gleich priorisierter CPUs aufgebaut sind, und in Systeme mit Co-Prozessoren für spezielle Aufgaben wie z. B. Arithmetik- oder Grafik-Prozessoren.

1.1 Festgekoppelte Mehrprozessor-Systeme

Festgekoppelte Systeme haben meist einen gemeinsamen Speicher und/oder eine gemeinsame Ein-/Ausgabe mit einem gemeinsamen Bus, während die einzelnen Prozessoren in diesem System zusätzlich ihren privaten Bus mit Speicher und Peripherie besitzen.

Festgekoppelte Z8000-Systeme können nach dem Master-Slave-Prinzip und nach dem Konzept gleichberechtigter CPUs aufgebaut sein. Für ein Master-Slave-System hat der Z8000 zwei spezielle CPU-Anschlüsse („Multi-Micro In" und „Multi-Micro Out") sowie vier Befehle: „Multi-Micro Request", „Multi-Micro Set", „Multi-Micro Reset" und „Multi-Micro Bit-Test".

Für ein System mit gleichberechtigten CPUs besitzt der Z8000 den „Test and Set"-Befehl, der nicht durch einen Bus-Request unterbrechbar ist und ein spezielles Statussignal (Z8003/4) ausgibt, das für eine „Bus-Lock"-Logik verwendet werden kann. Mit diesem „Test and Set"-Befehl kann z. B. ein Semaphore-Byte in einem gemeinsamen Speicher abgefragt und gesetzt werden.

1.2 Lose gekoppelte Mehrprozessor-Systeme

In einem lose gekoppelten Mehrprozessor-System arbeiten alle Prozessoren im System weitgehend unabhängig voneinander. Die Datenübergabe zwischen einzelnen Prozessoren geschieht über ein E/A-Port, eine spezielle Schnittstelle oder über einen Pufferspeicher (Dual-Port-RAM). Die Methode der Datenübergabe über einen intelligenten Pufferspeicher, wie z. B. dem FIO Z8038 von Zilog, ist eine sehr leistungsfähige und einfach zu realisierende Methode (*Bild 2*).

2 FIO Z8038 („FIFO Input/Output Interface Unit")

Der FIO-Baustein Z8038 ist ein asynchroner FIFO-Pufferspeicher für ein Interface zwischen CPU und CPU oder CPU und Peripherie.

Der interne FIFO-Pufferspeicher besitzt eine Kapazität von 128 × 8 Bit und kann in der Breite durch Parallelschalten mehrerer FIOs und in der Tiefe durch einen speziellen Erweiterungs-Baustein (Z8060 FIFO), erweitert werden (*Bild 3* und *4*).

An Hand der Blockschaltung (*Bild 1*) kann man erkennen, daß der FIO Z8038 völlig symmetrisch aufgebaut

Bild 1. Blockschaltung des FIO-Bausteins Z8038

ist. Port 1 und Port 2 sind die programmierbaren Schnittstellen, die ein elastisches Interface zwischen verschiedenen Mikrocomputern oder Mikrocomputer und Peripherie ermöglichen. Zum Beispiel könnte an Port 1 eine CPU Z8000 angeschlossen werden, die mit einem schnellen „Block-Move"-Befehl den internen FIFO-Puffer der FIO vollschreibt, während auf der anderen Seite an Port 2 ein langsames Peripheriegerät, z. B. ein Drucker, unabhängig von der Geschwindigkeit der CPU Byte für Byte aus dem Puffer ausliest (*Bild 6*).

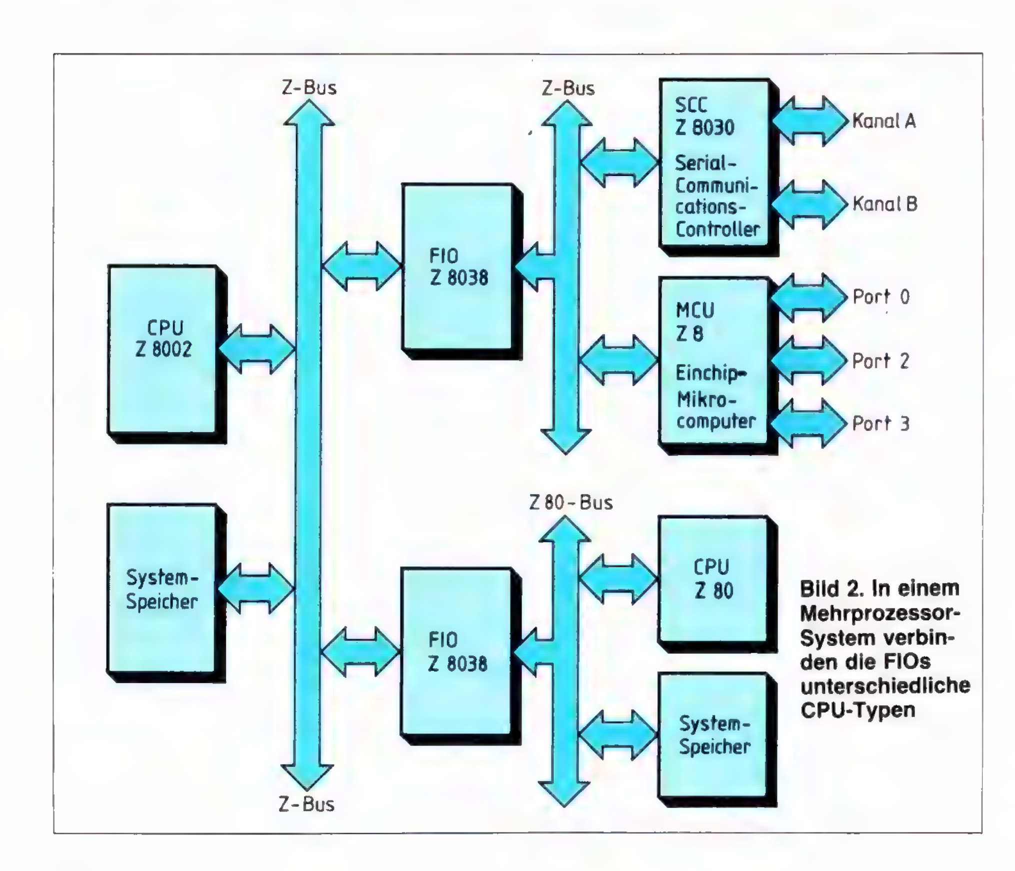


Bild 2. In einem Mehrprozessor-System verbinden die FIOs unterschiedliche CPU-Typen

Als weitere Anwendung könnte z. B. an Port 2 eine CPU Z8000A und an Port 1 eine CPU Z80B angeschlossen werden, die beide mit 6 MHz Taktfrequenz arbeiten und völlig asynchron über den FIO miteinander kommunizieren können.

Der FIO Z8038 unterstützt auch den direkten Speicherzugriff (DMA) für eine schnelle blockweise Datenübertragung von Port 1 oder Port 2 in den FIO-Speicher oder umgekehrt.

Port 1 und 2 kann für das Interface zwischen CPU und CPU unabhängig vom jeweils anderen Port als Schnittstelle im Zeitmultiplex (Adressen/Daten z. B. Z8, Z8000, 8085) oder als multiplexfreie Schnittstelle (Z80, 8080, 6800) programmiert werden.

An Port 1 muß dabei immer eine CPU angeschlossen werden, während Port 2 als Peripherie- oder ebenfalls als CPU-Interface dienen kann (*Tabelle 1*).

Der interne FIFO-Pufferspeicher ist als Dual-Port-RAM aufgebaut, der über einen Schreib- und Lesezähler adressiert wird. Diese Technik hat den Vorteil, daß jederzeit die Differenz zwischen den gelesenen und geschriebenen Bytes die Anzahl der gültigen Datenbytes im Speicher angibt. Die Differenz steht im „Byte-Count"-Register, das von beiden Ports gelesen werden kann.

Außerdem kann der FIO so programmiert werden, daß bei einer bestimmten, frei wählbaren Anzahl von Datenbytes im Puffer ein Interrupt oder ein DMA-Request ausgelöst wird.

Die „Pattern-Match"-Logik löst ebenfalls ein Interrupt-Request aus, falls ein gelesenes oder geschriebenes Byte mit dem Inhalt des „Pattern-Match"-Registers übereinstimmt. Somit kann man z. B. beim Übertragen eines Datenblocks mit dem End-of-File-(EOF) oder End-of-Text-Zeichen (EOT) einen Interrupt auslösen.

Das Message-Register, das in Wirklichkeit aus zwei Registern besteht (Schreiben/Lesen), dient dazu, in einem Mehrprozessor-System schnell eine Information von einem System an das andere zu übergeben. Bei der Übergabe wird gleichzeitig ein Interrupt ausgelöst, der dem anderen System das Vorliegen einer Message signalisiert.

Eine solche Information kann z. B. das „End of File"-Zeichen oder die Blocklänge eines Datenblocks sein.

2.1 *Interrupt-Struktur des FIO*

Die interne Interrupt-Struktur des FIO besitzt, wie alle Peripherie-Bausteine von Zilog, eine Prioritätskette (Daisy Chain). Durch die Position innerhalb der Kette wird die Priorität jeder Interrupt-Anforderung festgelegt. Der FIO hat pro Port sieben Interrupt-Quellen, Port 1 und 2 sind völlig

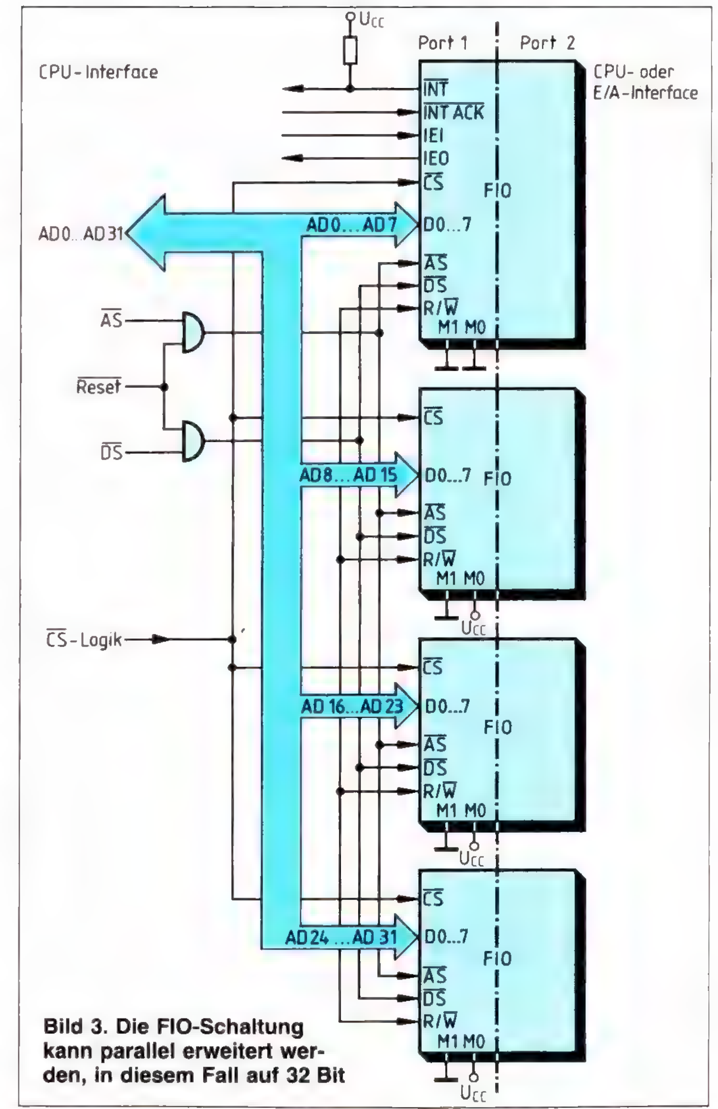


Bild 3. Die FIO-Schaltung kann parallel erweitert werden, in diesem Fall auf 32 Bit

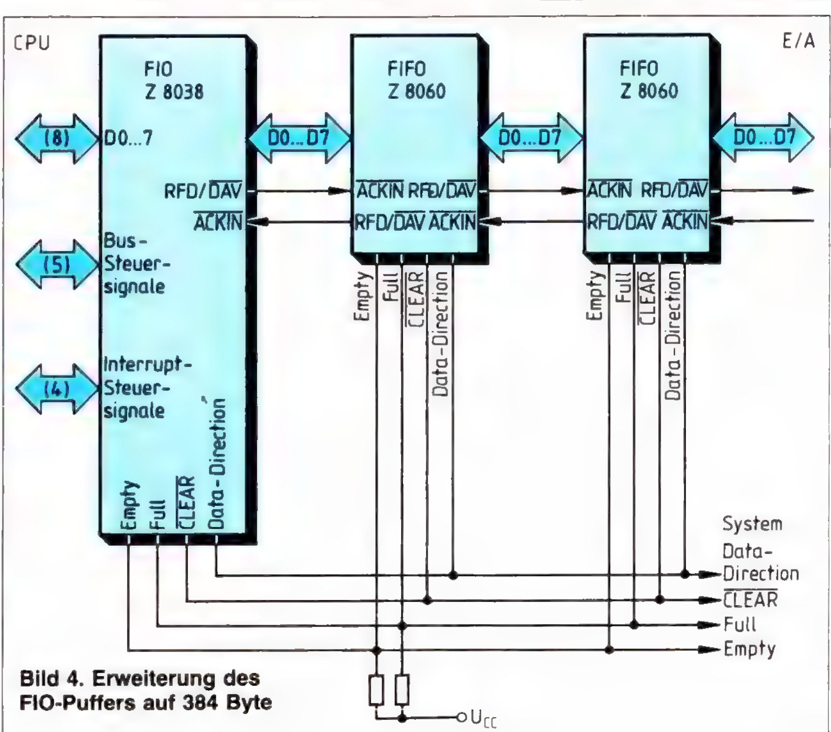


Bild 4. Erweiterung des FIO-Puffers auf 384 Byte

Tabelle 1. Anschlußmöglichkeiten von Port 1 und Port 2

Port 1 Programmierung (Hardware) M_1 M_0	Port 2 Programmierung (Software v. Port 1) B_1 B_0	Port 1 Anschlußbelegung	Port 2 Anschlußbelegung
0 0	0 0	Multiplex-Bus Low Byte	MPX-Bus, Low Byte
0 0	0 1		kein Multiplex
0 0	1 0		Drei-Draht-Handshake
0 0	1 1		Zwei-Draht-Handshake
0 1	0 0	Multiplex-Bus High Byte	MPX-Bus, High Byte
0 1	0 1		kein Multiplex
0 1	1 0		Drei-Draht-Handshake
0 1	1 1		Zwei-Draht-Handshake
1 0	0 0	direkter Bus	MPX-Bus, Low Byte
1 0	0 1		kein Multiplex
1 0	1 0		Drei-Draht-Handshake
1 0	1 1		Zwei-Draht-Handshake

symmetrisch aufgebaut (*Bild* 6). Wenn Port 1 und 2 jeweils an eine CPU angeschlossen sind, so kann jede Seite ihrer CPU bei einem der folgenden Gründe eine Interrupt-Anforderung liefern: Schreiben in das Message-Register (Interrupt-Anforderung an die jeweils andere CPU), ein Wechsel des Datenflusses durch den Puffer, eine Übereinstimmung eines Bit-Musters, das Erreichen eines bestimmten Byte-Zählerstandes, ein Puffer-Overflow- oder -Underflow-Fehler, Puffer voll oder Puffer leer.

Jede Interrupt-Quelle kann individuell freigegeben oder gesperrt werden und hat ihren eigenen Interrupt-Vektor. Dieser kann jedoch auch unterdrückt werden, so daß bei einem Interrupt-Acknowledge auch ein externes Gerät Daten auf den Bus legen kann.

2.2 FIO im DMA-Betrieb

Der FIO kann im Multiplex- und im direkten Betrieb mit einem DMA-Baustein zusammenarbeiten. Dabei unterstützt er zwei Arten der DMA-Übertragung: Das „Fly By" und das sequentielle Übertragen von Bytes. In der „Fly By"-Betriebsart wird bei jedem Maschinenzyklus ein Byte vom Speicher in den FIO oder umgekehrt übertragen. Bei der sequentiellen Übertragung wird während des ersten Maschinenzyklus das Byte vom Speicher oder der E/A-Einheit in den DMA-Baustein gelesen und beim nächsten Zyklus in den FIO geschrieben oder umgekehrt. Der DMA-Betrieb kann gleichzeitig auf beiden Ports des FIO stattfinden, der DMA-Baustein liefert dabei Speicheradressen und Read-/Write-Steuersignale, während der FIO Daten sendet oder empfängt. Ein spezielles Statussignal ($\overline{REQ}$) gibt im „Fly By" oder im sequentiellen Übertragungsmodus Auskunft über den Zustand des FIO-Puffers.

Wenn Daten in den FIO geschrieben werden, ist das $\overline{REQ}$-Signal aktiv, bis der Puffer voll ist. Das $\overline{REQ}$-Signal bleibt nun inaktiv, bis vom anderen Port der Puffer geleert wurde oder die Anzahl der Bytes im Puffer kleiner wird, als im „Bytecount"-Register angegeben wurde.

Sollen Daten aus dem FIO gelesen werden, ist das $\overline{REQ}$-Signal so lange inaktiv, bis vom anderen Port der Puffer vollgeschrieben wurde oder die im Byte-Count-Register angegebene Zahl überschritten wird. Das $\overline{REQ}$-Signal bleibt nun so lange aktiv, bis der DMA-Controller den Puffer geleert hat.

3 FIO-gekoppeltes Mehrprozessor-System

Besonders bei Mehrprozessor-Systemen (*Bild* 2) erweist sich eine Kopplung der einzelnen Systeme über ein Dual-Port-RAM wie dem FIO Z8038 als einfach zu realisierende und sehr effiziente Lösung. Die Busschnittstelle auf der Seite des Port 1 wird per Hardware über die beiden Eingänge M_1 und M_0 definiert, die Art der Schnittstelle von Port 2 muß von der an Port 1 angeschlossenen CPU per Befehl festgelegt werden (*Tabelle* 1). So kann z. B. an Port 1 ein Z8001-System angeschlossen werden und an Port 2 ein Z80-System oder an Port 1 ein Einchip-Mikrocomputer Z8 und an Port 2 ein Z8002-System. Die Datenübergabe zwischen den Systemen kann über zwei Wege geschehen: Erstens über den internen FIFO-Pufferspeicher und zweitens über das Message-Register, das den FIFO-Puffer umgeht (Bild 1). Zum vollständigen Initialisieren der FIO müssen die CPUs an beiden Ports die internen Steuerregister der FIO (16 auf jeder Seite) beschreiben. Ist eine CPU mit Multiplex-Adreß-/Daten-Bus angeschlossen, erfolgen Registeradressierung und Datentransfer in einem Zyklus. Bei direktem Bus-Interface ist der Zugriff zu den internen Registern eine Operation mit zwei Schritten. Mit dem ersten Ausgabe-Befehl der CPU wird ein Zeiger

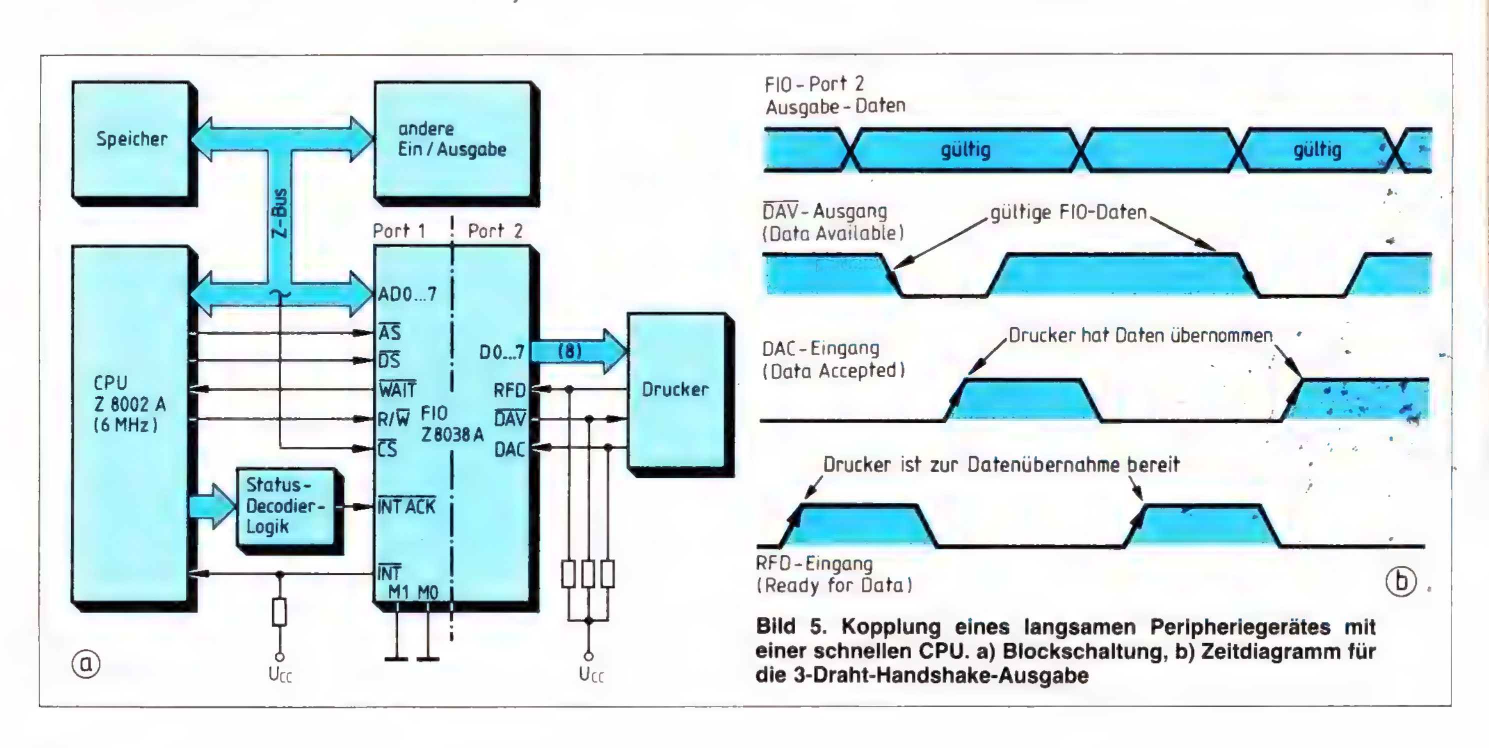


Bild 5. Kopplung eines langsamen Peripheriegerätes mit einer schnellen CPU. a) Blockschaltung, b) Zeitdiagramm für die 3-Draht-Handshake-Ausgabe

zu dem jeweiligen Register übergeben und mit dem zweiten das eigentliche Datum. Das Schreiben und Lesen des internen Pufferspeichers ist jedoch immer eine Ein-Schritt-Aktion.

In dem System nach Bild 3 kann z. B. der Z8002 mit einem schnellen Block-Move-Befehl den FIO-Puffer zum Z80-System beschreiben. Ist das Z80-System gerade noch nicht bereit, die Daten aus dem FIO abzuholen, wird ein „Buffer-Full"-Interrupt zum Z8002 den Datentransfer stoppen, bis die CPU Z80 den Puffer lesen kann.

Ist der Puffer leer, wird ein „Buffer-Empty"-Interrupt an die CPU Z80 geliefert. Über das Message-Register kann dem anderen System z. B. die Länge des Datenblocks, das Ende-Zeichen oder die Zieladresse des Datenblocks mitgeteilt werden.

4 Beispiel: CPU-Drucker-Interface

Bild 6a zeigt den Anschluß einer schnellen CPU Z8002A (6 MHz) an ein langsames Peripheriegerät (Drucker). Auf der CPU-Seite ist der FIO an den unteren Teil des 16-Bit-Busses (Z-Bus-Low) angeschlossen, während auf der anderen Seite der Drucker mit einem 3-Draht-Handshake-Interface an den FIO angeschlossen ist. Die CPU Z8002A kann nun z. B. mit einem Block Ausgabe-Befehl (OTIRB = „Output Byte Increment Repeat") mit einer Geschwindigkeit von max. 600 KByte pro Sekunde in den FIO-Puffer schreiben. Sobald das erste Byte im Puffer steht, zeigt das Signal $\overline{\text{DAV}}$ („Data Available") dem Drucker ein gültiges Datenbyte an. Signalisiert der Drucker mit seinen Signalen DAC („Data Accepted") und RFD („Ready For Data"), daß er das erste Byte übernehmen kann (Bild 6b), wird das erste Byte zum Drucker übertragen.

Da die CPU wesentlich schneller als der Drucker ist, wird der Puffer nach kurzer Zeit voll sein. Dabei werden die FIO-Signale $\overline{\text{WAIT}}$ und $\overline{\text{INTERRUPT REQUEST}}$ (für Puffer voll) aktiviert. Mit dem Interrupt-Request-Signal kann in eine Unterbrechungs-Routine verzweigt werden, die das Schreiben der CPU in den FIO-Puffer stoppt und das Bearbeiten anderer Programmteile startet. Hat der Drucker den Puffer ausgelesen, wird ein weiterer Interrupt-Request („Buffer Empty") geliefert, der das Schreiben von Daten in den FIO-Puffer erneut startet. Dadurch ergibt sich eine optimale Ausnutzung der Prozessorgeschwindigkeit.

5 Erweiterungsmöglichkeiten des FIO

Der FIO Z8038 ist dafür ausgelegt, daß er in der Tiefe und in der Breite erweitert werden kann.

Durch einfaches Parallelschalten von mehreren FIOs kann der 8-Bit-Datenweg auf 16, 24 oder 32 Bit erweitert werden (Bild 3). Durch Beschaltung der Anschlüsse M_0 und M_1 kann man festlegen, ob z. B. ein FIO an den unteren Teil des Z-Busses oder an den oberen Teil angeschlossen werden soll. Der FIO am Low-Teil des Z-Busses übernimmt dabei immer die Interrupt-Steuerung.

In der Tiefe kann der 128 Byte große FIO-Puffer durch eine Reihenschaltung mit einem FIO-Erweiterungsbaustein (FIFO Z8060) vergrößert werden.

Dieser 128 Byte große Erweiterungsbaustein ist ein nicht intelligenter FIFO-Puffer, der auch alleine ohne den FIO Z8038 für eine Ein/Ausgabe-Pufferung benutzt werden kann.

Soll ein FIO, der für eine CPU-CPU-Kopplung vorgesehen ist, erweitert werden, muß an jede CPU ein intelligenter FIO Z8038 angeschlossen werden.

Zwischen den beiden intelligenten FIOs können nun beliebige, nicht intelligente Erweiterungsbausteine Z8060 geschaltet werden. Bei einer FIO-Erweiterung, die für eine CPU-E/A-Kopplung vorgesehen ist, braucht nur die CPU-Serie einen intelligenten FIO Z8038, während auf der E/A-Seite FIOs Z8060 angeschlossen werden können (Bild 4).

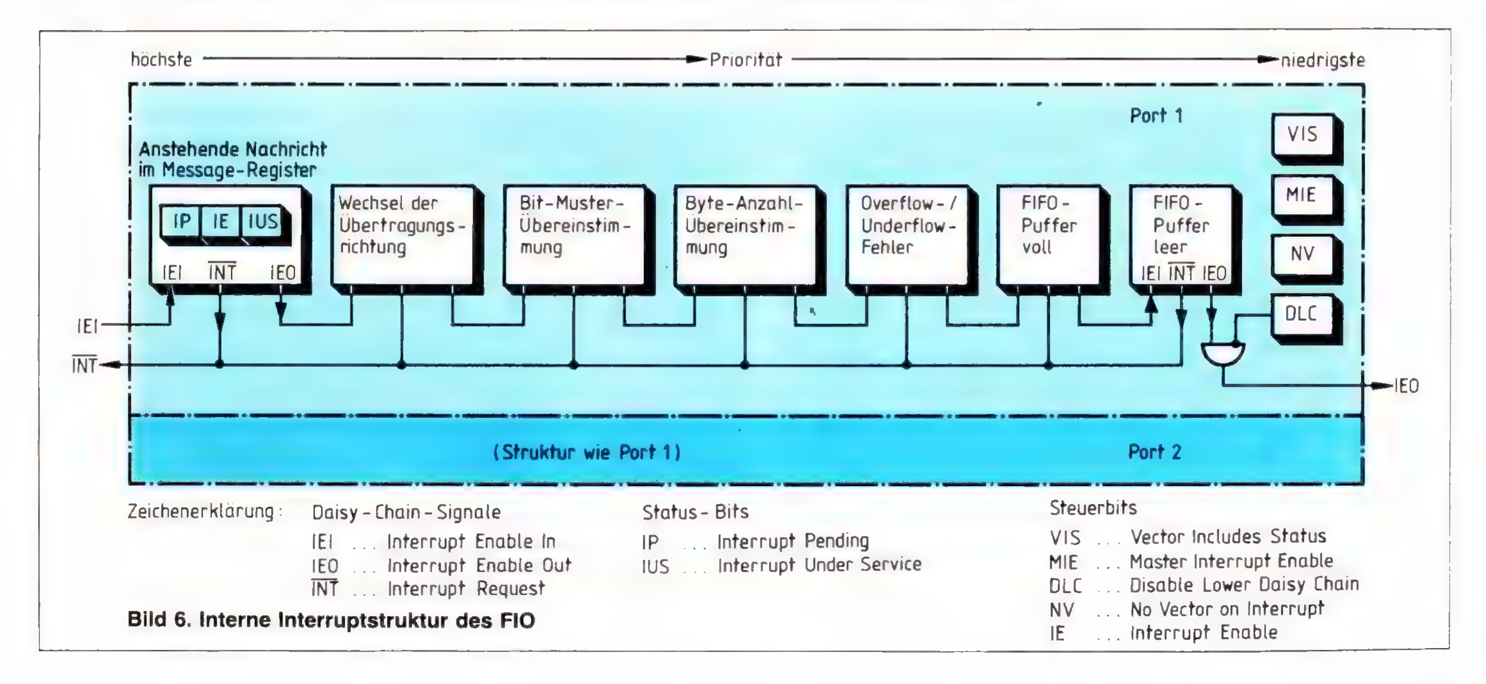


Bild 6. Interne Interruptstruktur des FIO

Dipl.-Ing. Werner Handke

Fehlersuche in Multi-Prozessor-Systemen

Bei der Fehlersuche in Mehrfach-Prozessor-Systemen wäre es die ideale Lösung, für jeden im System vorhandenen Mikroprozessor ein Entwicklungssystem vorzusehen. Diese Methode ist nicht nur redundant, sondern auch kostspielig. Die Benutzung bereits vorhandener Laborgeräte erweist sich als der praktischere Weg. Als Beispiel seien ein Entwicklungssystem und ein Logikanalysator genannt, die die notwendigen Debug-Hilfsmittel darstellen.

Anhand des Entwicklungssystems 8550 und des Logikanalysators 7D02 von Tektronix soll die Fehleranalyse für Mehrprozessorsysteme gezeigt werden. Beide Geräte sind direkt für externe Triggerung vorbereitet und stellen damit bereits das wichtigste Hilfsmittel für die Fehleranalyse zur Verfügung.

Während der Fehlersuche wird es selten notwendig, die Programme beider Mikroprozessoren gleichzeitig zu manipulieren. Im allgemeinen wird ein Prozessor unabhängig vom anderen soweit als möglich programmiert und ausgetestet. Danach wird der zweite Prozessor in Betrieb genommen und das Zusammenwirken zwischen beiden Systemen beobachtet. Nun wird es wichtig, beide Prozessoren bezüglich unsauberer oder fehlerhafter Operationen zu überwachen und Unterbrechungspunkte bzw. erzwungenes Anhalten für beide Systeme bereitzustellen. Nur auf diese Weise können fehlerhafte Daten und Programmausführung von beiden Systemen sorgfältig analysiert werden.

Im folgenden sind zwei Problemfälle beschrieben, die in Multi-Prozessor-Systemen typischerweise auftreten können. An einigen Beispielen wird das Zusammenwirken von Entwicklungssystem und Logikanalysator beschrieben, um die Fehleranalyse betreiben zu können.

Triggerung von Logikanalysator und Entwicklungssystem

Beim Zusammenwirken des Entwicklungssystems mit dem Logikanalysator gibt es zwei Arten zu triggern:

- entweder wird das Entwicklungssystem vom Logikanalysator getriggert
- oder das Entwicklungssystem triggert den Logikanalysator.

Zunächst wird die Methode des Triggerns des Logikanalysators vom Entwicklungssystem betrachtet.

Triggerung des Logikanalysators vom Entwicklungssystem

An der Rückseite des Entwicklungssystems 8550 befinden sich zwei Triggerausgänge für externe Geräte. Sie sind mit „EVENT 1" und „EVENT 2" bezeichnet (*Bild 1*). Falls eines der zugeordneten internen Triggerereignisse auftritt, wird am entsprechenden Ausgang ein positiver TTL-Impuls auftreten. Diese Ausgänge können entweder an die Triggereingänge (TRIGGER IN) oder an einen oder mehrere der Eingangskanäle des Logikanalysators angeschlossen werden (*Bild 2*). Mit der beschriebenen Zusammenschaltung kann der Logikanalysator bezüglich der vielfältigen Trigger-Kombinationen des Entwicklungssystems getriggert werden.

Entwicklungssystem vom Logikanalysator getriggert

Um das Entwicklungssystem mit dem Logikanalysator zu triggern, wird der Triggerausgang (TRIGGER OUT) des Logikanalysators mit irgendeinem Eingangskanal des Achtkanal-Tastkopfs des Entwicklungssystems verbunden (*Bild 3*). Das Entwicklungssystem kann eingestellt werden, um auf einen oder mehrere Triggerpulse zu reagieren. Zusätzlich sind logische Verknüpfungen des Triggerpulses mit dem Auftreten anderer Ereignisse möglich.

Ergänzend besteht an beiden Systemen die Möglichkeit, komplexe Triggerbedingungen zu spezifizieren, die bei der Fehlersuche in komplizierten Fällen erfolgreich sind. Als Beispiel sei der repetierende Fall genannt: Der Logikanalysator triggert das Entwicklungssystem, welches wiederum den Logikanalysator triggert usw.

Beispiel für eine Prozessorschaltung

Das in *Bild 4* gezeigte Schaltungsbeispiel wird für die folgende Fehleranalyse herangezogen. Es enthält

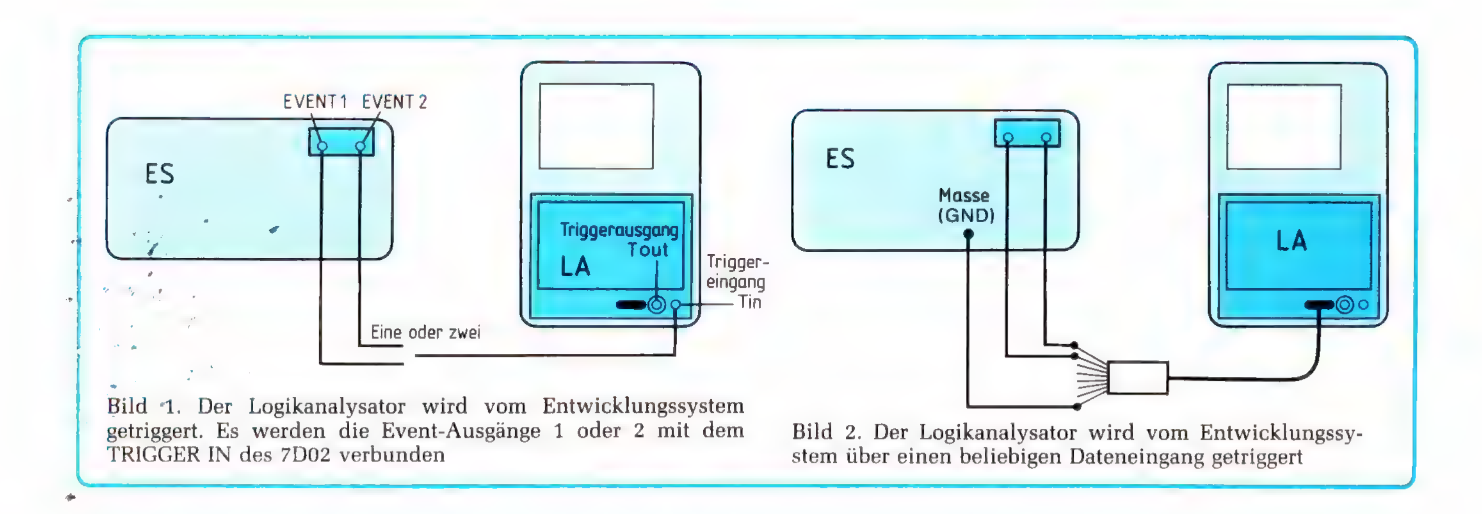


Bild 1. Der Logikanalysator wird vom Entwicklungssystem getriggert. Es werden die Event-Ausgänge 1 oder 2 mit dem TRIGGER IN des 7D02 verbunden

Bild 2. Der Logikanalysator wird vom Entwicklungssystem über einen beliebigen Dateneingang getriggert

zwei Prozessoren Z80, beide mit eigenem ROM und RAM. Zur Vereinfachung seien identische Adreßbereiche angenommen.

ROM 0000...3FFF (alle Adressen sedezimal)
RAM 4000...4FFF.

Zusätzlich haben beide Prozessoren zum gemeinsamen RAM im Bereich 5000...5FFF Zugriff. Prozessor 1 bearbeitet die E/A-Funktionen über die Peripherie-Baugruppe 1, während Prozessor 2 alle Probleme der Datenmanipulation durchführt und sich zur Massenspeicherung der Peripherie-Baugruppe 2 bedient. In diesem Schaltungsbeispiel wird davon ausgegangen, daß die Prozessoren asynchron über das gemeinsame RAM kommunizieren, was durch entsprechende Datenworte initiiert wird. Damit sind vielfältige Fehlermöglichkeiten gegeben.

Problemstellung 1

Während eines Datentransfers von Prozessor 1 zu Prozessor 2 gehen einige Byte verloren, wodurch Systemfehler hervorgerufen werden. Prozessor 1 überträgt beispielsweise 16...128 Byte in den gemeinsamen RAM-Bereich und veranlaßt Prozessor 2 durch Interrupt die gespeicherten Daten zu lesen. Die Anzahl der gelesenen Daten ist aber geringer als die der eingeschriebenen. Wodurch wird nun der Fehler verursacht? Durch den Prozessor 1, das RAM oder den Prozessor 2?

Lösung

Um dieses Problem zu lösen, werden folgende Informationen benötigt:

– Wieviele Daten wurden gesendet?
– Wie sahen die Daten aus?
– Wieviele Daten wurden empfangen?
– Wie sahen die Daten aus?

Steht für die Problemlösung nur ein Entwicklungssystem zur Verfügung, wird es ziemlich schwierig sein und mehrere Durchläufe erfordern, zuerst den einen Prozessor zu emulieren und dann das Verhalten des anderen durch Emulation zu analysieren. Die Identität der zu vergleichenden Durchläufe wäre dabei nicht gewährleistet. Eine Methode zur Fehlersuche wäre, beide Systeme einzusetzen: Mit dem Entwicklungssystem wird das Schreiben der Daten in das RAM überwacht und aufgezeichnet. Danach wird der Logikanalysator getriggert und die Aufzeichnung der gelesenen Daten getestet. Die Systemeinstellungen werden im folgenden beschrieben.

Einstellung

Weil das Entwicklungssystem den Logikanalysator triggern muß, werden die Systeme, wie in Bild 1 gezeigt, miteinander verbunden. Der Ausgang EVT2 wird an den Eingang TRIGGER-IN (extern) des Logikanalysators angeschlossen.

Jetzt muß ein Ereignispunkt des Entwicklungssystems so gesetzt werden, daß auf jedes Schreiben in den gemeinsamen RAM-Bereich ein Trigger ausgelöst wird. Dieser Trigger wird als Zählimpuls für einen Mehrzweckzähler verwendet.

> EVT 1 CLR A > 4FFF B=MW

Ereignispunkt 2 muß so definiert werden, daß das Ende des Schreibens von Daten erkannt wird. Wird das Schreiben beispielsweise mit dem Befehl RET (C9) an Adresse 234C beendet, so lautet die Einstellung für Ereignispunkt 2:

> EVT 2 CLR A = 234CB = FD = C9

Der Befehl zum Abbrechen der Programmausführung muß mit dem Beenden des Schreibens in den RAM-Bereich aktiv werden.

> BIF ARM

Der Mehrzweckzähler soll die Ereignisse 1 zählen und zeigt damit die Anzahl der geschriebenen Byte:

> CNT 1

Schließlich wird für den Echtzeit-Speicher des Entwicklungssystems definiert, nur das Schreiben von Daten in das RAM aufzuzeichnen. Es können dann maximal 128 Byte aufgezeichnet werden.

> RTT MW

Das Entwicklungssystem ist jetzt bereit, festzustellen, wieviele Byte zum Prozessor 2 gesendet worden sind, diese in seinem Echtzeit-Speicher aufzuzeichnen und bei Erreichen von Ereignispunkt 2 die Programmausführung abzubrechen. Der Logikanalysator muß noch eingestellt werden, damit er feststellt, wie-

viele Daten von Prozessor 2 gelesen werden, und muß die Daten in seinem Speicher aufzeichnen. Nach dem Lesen aller Daten muß Prozessor 2 angehalten werden, um die gespeicherten Daten und den Inhalt des gemeinsamen RAM-Bereichs darzustellen. Zunächst muß der Analysator 7D02 mit Test 1 so eingestellt werden, daß er auf den Triggerimpuls EVT2 vom Entwicklungssystem wartet. Beim Empfangen des Triggers springt der Logikanalysator nach Test 2.

```
TEST 1
1IF
1 WORD RECOGNIZER #1
1 DATA=XX
1 ADDRESS=XXXX
1 R/W=X FETCH=X IO/MEM=X INT=X
1 /INTAK=X EXT TRIG IN=1
1 TIMING WR=X
1THEN DO
1 GO TO 2
END TEST 1
```

Test 2 überwacht jedes Lesen aus dem gemeinsamen Speicherbereich und inkrementiert bei jedem Lesen Zähler 1. Die Speicherauswahl wird gesetzt, um alle Daten beim Lesen aus dem gemeinsamen RAM in den 256-Worte-Speicher des Logikanalysators einzulesen. Als Teil von Test 2 wird der Worterkenner 3 gesetzt, daß er auf das Ende des Lesens aus dem RAM triggert, worauf der Logikanalysator den Prozessor anhält und die gespeicherten Informationen anzeigt:

```
TEST 2
2IF
2 WORD RECOGNIZER #2
2 DATA=XX
2 ADDRESS=5XXX
2 R/W=1 FETCH=X IO/MEM=0 INT=X
2 /INTAK=X EXT TRIG IN=X
2 TIMING WR=X
2 THEN DO
2 COUNTNER # 1 0-EVENTS
2      0-INCREMENT
2 OR IF
2 WORD RECOGNIZER #3
2 DATA=C9
2 ADDRESS=2A43
2 R/W=X FETCH=1 IO/MEM=X INT=X
2 /INTAK=X EXT TRIG IN=X
2 TIMING WR=X
2 TRIGGER
2      3-ZERO DELAY
2      1-SYSTEM UNDER TEST HALT
2      0-STANDARD CLOCK QUAL.
END TEST 2
QUALIFY
Q STORE ON
Q WORD RECOGNIZER #2
Q DATA=XX
Q ADDRESS=5XXX
Q R/W=1 FETCH=X IO/MEM=0 INT=X
Q /INTAK_X EXT TRIG IN=X
Q TIMING WR=X
END QUALIFY
```

An dieser Stelle sind beide Prozessoren angehalten worden. Der Inhalt des Echtzeit-Speichers enthält die letzten 128 Byte, die in den gemeinsamen RAM-Bereich geschrieben worden sind, der Mehrzweckzähler enthält die Anzahl der gesendeten Byte. Der Speicher des Logikanalysators enthält alle gelesenen Byte aus dem gemeinsamen RAM und der Zähler 1 enthält die Anzahl der gelesenen Byte. Diese Informationen können nun ausgewertet werden, um die Zahl der verlorenen Byte festzustellen und welche Byte es waren. Damit wird die Art des Fehlers offensichtlich oder es können Bedingungen gefunden werden, die den Fehler verursachen, und deren Ursache kann weiterverfolgt werden.

Problemstellung 2

Prozessor 1 empfängt Daten von der Peripheriebaugruppe 1. Abhängig von diesen Daten werden vom Prozessor 1 verschiedene Aufgaben an Prozessor 2 gestellt. Während einer bestimmten Folge zweier Anfragen werden von Prozessor 2 fehlerhafte Daten an Prozessor 1 übertragen, wodurch Prozessor 1 außer Kontrolle gerät. Das Übertragen von Aufgaben an Prozessor 2 erfolgt durch Senden eines eindeutigen Befehlswortes zusammen mit den benötigten Daten. In der fehlerhaften Sequenz überträgt Prozessor 1 zuerst 2C und einige Datenbyte an Prozessor 2. Danach übergibt Prozessor 1 das Befehlswort D1 mit dazugehörenden Daten. Während der Bearbeitung der zweiten Aufgabe tritt ein Fehler auf und Prozessor 2 gibt das fehlerhafte Datenpaket an Prozessor 1. Der Fehler tritt allerdings nur auf, wenn die beiden beschriebenen Aufgaben nacheinander ausgeführt werden. Da das Aufeinanderfolgen beider Aufgaben zufällig auftritt und nur mit aufwendigen Hilfsprogrammen reproduzierbar ist, wird die Fehlersuche schwierig sein.

Lösung

Der Logikanalysator wird so eingestellt, daß er die Folge der beiden beschriebenen Aufgaben feststellt; dazu wird seine programmierbare Testmöglichkeit benutzt. Wenn dieses Ereignis eintritt, soll der Logikanalysator die Daten der Ausführung des zweiten

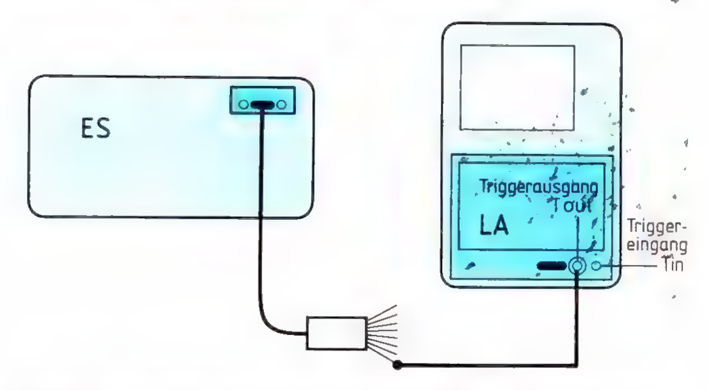


Bild 3. Das Entwicklungssystem 8550 wird vom Logikanalysator getriggert. TRIGGER OUT wird mit einem der Dateneingänge des Achtkanal-Testkopfs des 8550 verbunden

Programms aufzeichnen, und auf das Schreiben des fehlerhaften Datenpakets in das gemeinsame RAM warten. Damit wird das Entwicklungssystem, das die geschriebenen Daten aufzeichnet, getriggert. Auf diese Weise werden die Programmausführung und Ergebnisdaten darstellbar und damit die notwendige Information geliefert, um den Fehler zu finden.

Einstellungen

Damit der Logikanalysator das Entwicklungssystem triggern kann, müssen die Systeme so zusammengeschaltet werden, wie es in Bild 3 dargestellt ist. Der externe Triggerausgang des Logikanalysators wird an den ersten Kanal des Logiktastkopfes des Entwicklungssystems angeschlossen. Der Analysator 7D02, der für dieses Anwendungsbeispiel benutzt werden soll, ist nicht mit einer Zeitgeberfunktion (Timing-Option) ausgerüstet, deshalb gibt der externe Triggerausgang immer ein Triggersignal ab, wenn eine Triggerbedingung erfüllt ist.

Um dieses Problem zu lösen, muß der Logikanalysator so eingestellt werden, daß er die fehlerhafte Befehlsfolge erkennt. Weil diese Anforderungen in dem gemeinsamen RAM-Bereich nicht unter festen Adressen abgelegt werden, ist ihre Identifizierung beim Lesen fast unmöglich. Die Anforderungen werden allerdings vor ihrer Bearbeitung in einem festgelegten Speicherbereich des internen RAM zwischengespeichert. Deshalb kann dieser Speicherbereich beobachtet werden, und das Auftreten der Befehlsfolgen kann bei deren Einschreiben in den Speicher erkannt werden.

Um dies mit dem Logikanalysator durchzuführen, muß Test 1 die erste Anforderung (Befehl 2C) erkennen. Diese Anforderung wird an Adresse 4500 zwischengespeichert.

```
1 IF TEST 1
1 WORD RECOGNIZER #1
1 DATA=2C
1 ADDRESS=4500
1 R/W = 0 FETCH=X IO/MEM=0 INT=X
1 /INTAK_=EXT TRIG IN=X
1 THEN DO
1 GO TO 2
  END TEST 1
```

Test 2 erkennt, wenn in den Zwischenspeicher an Adresse 4500 das Byte D1 geschrieben wird. Falls das geschieht, springt das System zu Test 3, wodurch das Aufzeichnen von Daten beginnen wird. Es muß zusätzlich der Fall betrachtet werden, wo zwei aufeinanderfolgende 2C-Anforderungen auftreten. Falls dies eintritt, muß das System in Test 2 bleiben und nicht nach Test 1 zurückspringen. Sollte allerdings das nächste Datenwort, geschrieben an Adresse 4500, nicht D1 oder 2C sein, muß das System nach Test 1 zurückspringen.

```
TEST 2
2IF
2 WORD RECOGNIZER #2
2 DATA=D1
2 ADDRESS=4500
2 R/W=0 FETCH=X IO/MEM=0 INT=X
2 /INTAK=X EXT TRIG IN=X
2THEN DO
2 GO TO 3
2 OR IF
2 WORD RECOGNIZER #
2 DATA=2C
2 ADDRESS=4500
2 R/W=0 FETCH=X IO/MEM=0 INT=X
2 /INTAK=X EXT TRIG IN=X
2THEN DO
2 GO TO 2
2OR IF
2 WORD RECOGNIZER #3
2 DATA=XX
2 ADDRESS=4500
2 R/W=0 FETCH=X IO/MEM=0 INT=X
2 /INTAK=X EXT TRIG IN=X
2THEN
2 GO TO 1
  END TEST 2
```

Test 3 veranlaßt den Logikanalysator die nächsten 256 Befehle aufzuzeichnen, die von Prozessor 2 ausgeführt werden. Die Aufzeichnung wird beendet, wenn das mit dem Worterkenner 3 bezeichnete Wort auftritt. Falls vorher die 256-Byte-Grenze erreicht wird, springt das System nach Test 4, wo auf das Ende des angeforderten Programms gewartet wird, bevor das Entwicklungssystem getriggert wird.

```
TEST 3
3IF
3 COUNTER #1 = 256 0-EVENTS
3THEN
3 GO TO 4
3OR IF
3 WORD RECOGNIZER #4
3 DATA=C9
3 ADDRESS=2E20
3 R/W=X FETCH=1 IO/MEM=X INT=X
3 /INTAK=X EXT TRIG IN=X
3THEN
3 TRIGGER
3     3-ZERO DELAY
3     0-SYSTEM UNDER TEST CONT.
3     0-STANDARD CLOCK QUAL.
3ELSE
3
3 COUNTER # 1 0-EVENTS
3     0-INCREMENT
3 QUALIFY
3
END TEST 3
```

Mit Test 4 erwartet der Worterkenner 4 den Return-Befehl auf Adresse 2E20, womit das Ende der Routine angezeigt wird, bevor das Schreiben im gemeinsamen RAM beginnt. Nachdem alle Daten ins RAM geschrieben sind, triggert der Logikanalysator das Entwick-

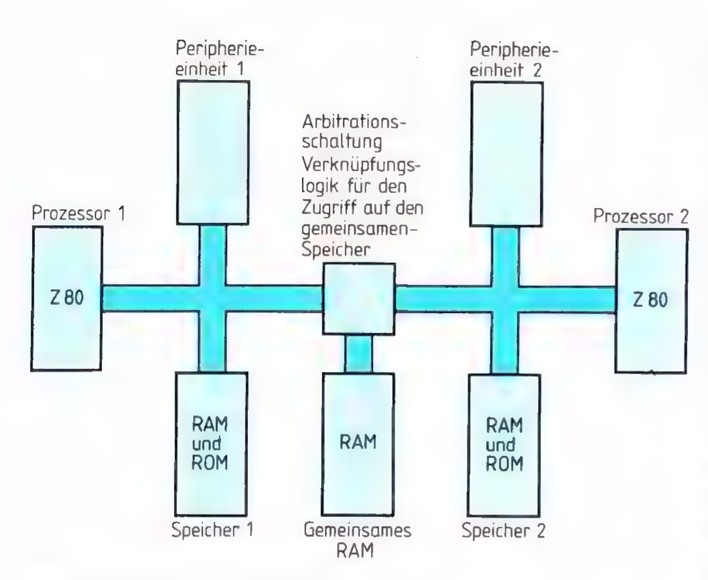


Bild 4. Blockschaltung des Testaufbaus

lungssystem, um die Daten aufzeichnen zu können, die nun aus dem gemeinsamen RAM gelesen werden.

```
TEST 4
4IF
4 WORD RECOGNIZER #4
4 DATA=C9
4 ADDRESS=2E20
4 R/W=X FETCH=1 IO/MEM=X INT=X
4 /INTAK=X EXT TRIG IN=X
4THEN DO
4 TRIGGER
4      3-ZERO DELAY
4      0-SYSTEM UNDER TEST CONT.
4      0-STANDARD CLOCK QUAL.
  END TEST 4
```

Das Entwicklungssystem muß auf die vom Logikanalysator gesendeten Trigger reagieren, um die von Prozessor 2 gesendeten Daten aufzeichnen zu können. Dazu wird EVENT 1 gesetzt, um das Triggersignal zu erkennen. Das Triggersignal wird am Kanal 1 des externen Tastkopfes am Entwicklungssystem eingespeist.

> EVT 1 CLR T=1xxxxxxx

EVENT 2 muß so eingestellt werden, daß Prozessor 1 angehalten wird, wenn das Lesen aus dem gemeinsamen RAM beendet wird. Die Lese-Routine endet, wenn an Adresse 2EC4 der Return-Befehl steht:

> EVT 2 CLR A=2EC4 D=C9 B=F

Um den Prozessor 1 mit EVENT 2 anzuhalten, wird der BREAK-IF-Befehl benutzt. Weil weiterhin Prozessor 1 nur angehalten werden soll, wenn EVENT 2 nach EVENT 1 auftritt, wird die ARM-Bedingung erforderlich.

> BIF ARM

Wie Prozessor 2 schreibt der Prozessor 1 alle vom gemeinsamen RAM gelesenen Daten in einen internen Zwischenspeicher. Deshalb wird das Entwicklungssystem veranlaßt, nur das Schreiben zum Speicher (Memory Writes) aufzuzeichnen. Damit werden die 128 letzten Byte erfaßt, die von Prozessor 2 gesendet worden sind.

> RTT MW

Wenn das gesamte Programm ausgeführt wird, wird nach Ausführung der Befehlsfolgen, die durch die Anforderungen 2C und D1 aufgerufen wurden, der Logikanalysator das Entwicklungssystem triggern und selbst die letzten 256 Befehle der nachfolgenden Routine aufzeichnen. Das Entwicklungssystem zeichnet die 128 Byte auf, die vom Prozessor 2 gesendet werden. Durch die Anwendung von Entwicklungssystem und Logikanalysator in gemeinsamem Einsatz können signifikante Daten von beiden Prozessoren aufgezeichnet werden und damit können Systemfehler gefunden werden. Wird nur eines der Hilfsmittel eingesetzt, und korrespondierende Daten in verschiedenen Meßdurchgängen ermittelt, besteht keine Sicherheit, daß in den verschiedenen Meßdurchgängen identische Fehler analysiert werden.

Mit den beschriebenen Meßmethoden können mit Entwicklungssystem und Logikanalysator, die sich wie das 8550 und der 7D02 ergänzen, komplizierteste Vorgänge in Multiprozessor-Schaltkreisen untersucht werden. Es gibt noch eine Reihe weiterer Anwendungen. Ein Beispiel wäre, mit dem Entwicklungssystem den Logikanalysator zu triggern und asynchron zeitkritisches Zusammenwirken von Baugruppen zu testen.

Literatur

[1] *Johnson, D.:* Debugging Multiprocessor Circuits Using Development Lab and a Logic Analyzer. Tektronix, Inc., Beaverton.

[2] DOS/50 8550 Microcomputer Development Lab, System Reference Booklet, Tektronix Inc. (Deutsche Überarbeitung: *W. Handke*).

[3] 8550 MDL System Users Manual, Tektronix, Inc.

[4] 7D02 Logic Analyzer Operators Manual, Tektronix, Inc.

[5] *Handke, W.:* Mikroprozessoren-Hilfsmittel zur Programmentwicklung und Testmethoden. Taschenrechner und Mikrocomputer Jahrbuch 1980, Vieweg Verlag.

[6] *Handke, W.:* Echtzeitanalyse als Hilfsmittel beim Mikroprozessor Schaltungsentwurf. Elektronik Information 80, H. 7, H. 8.

Die deutsche Überarbeitung dieses Beitrages erfolgte mit freundlicher Genehmigung von *D. Johnson*, Tektronik Inc., Beaverton.

Dipl.-Ing. Werner Handke, geb. in Berlin, studierte von 1971...77 an der TH Darmstadt Elektrotechnik/Nachrichtentechnik. Nach dem Studienabschluß trat er bei Rohde & Schwarz Engineering and Sales zunächst als Produktingenieur für Mikroprozessor-Entwicklungssysteme ein. Er war dort seit Ende 1978 als Verkaufsleiter für Mikroprozessor-Entwicklungssysteme und rechnergesteuerte Signalerfassungssysteme von Tektronix verantwortlich.
Hobbys: Musik hören, Modelleisenbahnbau und Mikroprozessoren.
Privattelefon: (02 21) 72 66 11.



Dr.-Ing. Werner Hoffmann

Variables Testkonzept für Mikroprozessorsysteme

Immer mehr Mikroprozessoren werden in mittleren oder großen Rechensystemen eingesetzt. Dabei wird es für den Benutzer immer wichtiger, fast zu jedem Zeitpunkt zu wissen, ob das System noch korrekt arbeitet. Da eine On-line-Fehlererkennung im laufenden Betrieb sehr aufwendig ist, begnügt man sich in unkritischen Anwendungen damit, den Rechner nach dem Systemstart, in festen Zeitintervallen oder in Leerzeiten zu testen. Nur die Datenwege und der Speicher werden on-line durch fehlererkennende (oder -korrigierende) Codes (Parity, Hamming usw.) überprüft. – Im folgenden wird ein Konzept für ein variables Diagnosesystem in Ein- und Mehrprozessorsystemen vorgestellt. Dabei wird angenommen, daß das Rechensystem aus einem oder mehreren Mikroprozessoren des gleichen Typs aufgebaut ist. Die Prozessoren sind durch einen gemeinsamen Systembus miteinander verbunden. Die Prozessoren haben eine Wortlänge von 16 oder 32 Bit und sind mikroprogrammiert.

1 Konzept des Testprozessors

Das Testsystem sieht vor, auf jedem Prozessorchip einen zusätzlichen Testprozessor zu integrieren. In Einprozessorsystemen übernimmt er die Steuerung des gesamten Systemtests. In Mehrprozessorsystemen wird ein Prozessor als Serviceprozessor ausgelegt. Dieser ist mit den Testprozessoren in den anderen Funktionsprozessoren über ein eigenes Kommunikationsnetz verbunden (*Bild 1*). Dadurch entsteht ein mächtiges hierarchisches Testsystem.

Der Testprozessor auf dem Chip übt sowohl reine Schaltfunktionen als auch Kontrollfunktionen aus. Der Serviceprozessor in einem Multiprozessorsystem übt die gleichen Funktionen aus, wie sie heute bei entsprechenden Prozessoren in Großrechnern üblich sind. Mit dem Testprozessor auf dem Prozessorchip und dem Serviceprozessor läßt sich ein dreistufiges Testkonzept verwirklichen:

- Auf der untersten Ebene testet sich der Testprozessor so gut wie möglich selbst und testet den Testkern im Funktionsprozessor, der für einen Prozessor-Selbsttest im Bootstrapping-Verfahren notwendig ist [1, 2]. Um einige Komponenten des Testkerns zu testen, verwendet der Testprozessor auch spezielle Testschaltungen (siehe Abschnitt 3).
- Auf der zweiten Ebene testet sich jeder Funktionsprozessor selbst. Dieser Selbsttest mit Hilfe von Mikroprogrammen wird nach dem Bootstrapping-Verfahren durchgeführt. Die Kontrolle über den Selbsttest und die Auswertung der Testergebnisse übernimmt der zugehörige Testprozessor.
- Auf der dritten Ebene übernimmt der Serviceprozessor in Multiprozessorsystemen die Koordination für einen Gesamttest. Er testet das Kommunikationsnetz zwischen den Testprozessoren (damit auch jeden einzelnen Testprozessor selbst) und die Komponenten des Systems, die sich nicht selbst testen können (Hauptspeicher, System-Bus usw.).

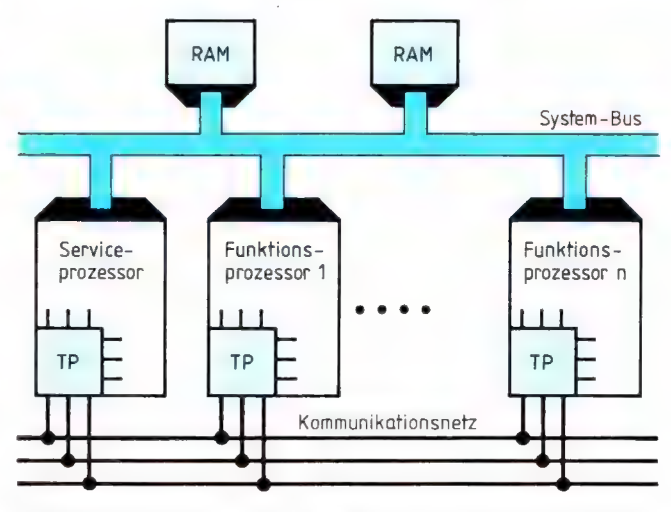


Bild 1. Testprozessoren und Serviceprozessor in einem Mehrprozessorsystem (TP = Testprozessor)

Im Falle eines Fehlers kann er zusammen mit dem Betriebssystem bei entsprechend vorhandener Redundanz eine Rekonfiguration durchführen.

2 Funktionen des Test- und des Serviceprozessors

Der Testprozessor auf dem Chip übt sowohl reine Schaltfunktionen als auch Steueraufgaben aus. Die wesentlichen Funktionen davon sind:

- Initialisierungs- und Wartungsfunktionen:
  Der Testprozessor macht nichtzugängliche Teile des Prozessors (Register, Statusflags usw.) von außen zugänglich. Damit erlaubt er, einen definierten Anfangszustand im Prozessor zu erzeugen, den Prozessor einschließlich des Mikrocontrollers in einen bestimmten Zustand zu bringen und sämtliche Register und Flags einzeln zu setzen und zu lesen.
- Kontrollfunktionen:
  Er überwacht die Mikroprogrammadressen und die Ausführungszeiten von Diagnoseprogrammen, die auf dem Prozessor ablaufen. Er wertet die Meldungen spezieller Prüflogiken aus, welche on-line die externen Datenwege und den Hauptspeicher überwachen und reagiert auf Fehler z. B. durch Anstoßen einer Befehlswiederholung oder mit einem Prozessor-Stopp.
- Ausführen und Steuern von Prüfprogrammen:
  Der Testprozessor testet sich in einem Grundtest soweit wie möglich selbst sowie den Testkern des Funktionsprozessors. Außerdem übernimmt er die Auswertung der Ergebnisse des Prozessor-Selbsttests.

Der Serviceprozessor übt Funktionen aus, wie sie heute bei entsprechenden Prozessoren in Großrechnern üblich sind. Dazu gehören:

- Systeminitialisierung:
  Zunächst wird vom Serviceprozessor eine Grundprüfung aller Komponenten des Gesamtsystems durchgeführt. Im Fehlerfalle erfolgt gegebenenfalls eine Rekonfiguration. Anschließend wird das Laden des Betriebssystems angestoßen.
- Unterstützen des Betriebssystems im Falle einer Störung:
  Der Serviceprozessor unterstützt das Betriebssystem im Falle einer durch Befehlswiederholung bzw. Speicherkorrektur nicht zu beseitigenden Störung bei der Rekonfiguration, wenn entsprechende Redundanz vorhanden ist.
- Ausführen von Prüfprogrammen:
  Der Serviceprozessor führt Prüfprogramme für den Arbeitsspeicher, das Unterbrechungssystem und andere Komponenten aus, die sich nicht selbst testen können. Er stößt den Testvorgang in den anderen Funktionsprozessoren an und wertet die Ergebnisse aus.
- Fehleranalyse:
  Im Serviceprozessor wird der von ihm bei Auftreten eines sporadischen oder festen Fehlers gesicherte Zustand einer Systemkomponente ausgewertet und – soweit notwendig – eine Fehlerlokalisierung durchgeführt. Außerdem führt er Fehlerstatistiken.
- Bereitstellen von Wartungsfeldroutinen:
  Der Serviceprozessor soll das herkömmliche Wartungsfeld bei Großrechnern ersetzen. Die Wartungsfeld-Zugriffe (z. B. Anzeige von Speicherinhalten, Anzeige und Setzen von Registern in den einzelnen Prozessoren, Einzelbefehlsausführung usw.) werden von Serviceprozessor-Routinen durchgeführt.
- Konsolbedienung:
  Der Serviceprozessor übernimmt die Aufgabe der Konsolsteuerung und bedient – transparent für das Betriebssystem – die Konsole. Dabei steuert er den Datenverkehr.
- Weitere Serviceprozessor-Dienste:
  Der Serviceprozessor sollte zusätzlich einige Dienstprogramme anbieten, z. B. für Speicherabzug usw.

3 Testprozessor und Prozessor-„Selbsttest“

Ein wesentlicher Teil im Testkonzept ist der Prozessor-Selbsttest mit Hilfe von Mikroprogrammen und zusätzlicher Testlogik nach dem Bootstrapping-Verfahren. Bei einem Selbsttest nach dem Bootstrapping-Verfahren [1, 2] wird zunächst der gesamte Prozessor in einzelne Testobjekte zerlegt und ein Testkern definiert. Unter dem Testkern werden alle die Komponenten des Prozessors verstanden, die zum Testen des ersten Testobjektes notwendig sind. Er soll möglichst minimal sein, da seine Komponenten als fehlerfrei vorausgesetzt bzw. von „außen“ getestet werden müssen. Ausgehend von diesem Testkern werden Schritt für Schritt die übrigen Testobjekte überprüft. Zum Überprüfen einer Komponente werden jeweils nur im bisherigen Ablauf getestete Komponenten benutzt.

Bei einem Selbsttest mit Hilfe von Mikroprogrammen besteht der Testkern aus dem Steuerwerk, d. h. der Mikroprogramm-Kontrolleinheit, dem Mikroprogrammspeicher und dem Pipeline-Register. Hinzu kommen noch das Makro-Befehlsregister und das „Mapping-PROM“, da diese beiden Komponenten nicht vom Prozessor selbst mit Mikroprogrammen getestet werden können (*Bild 2*). Dieser Testkern soll nun als erstes mit Hilfe des Testprozessors und zusätzlicher Testlogik für eine Signaturanalyse überprüft werden. Anschließend kann sich der Prozessor vollständig selbst testen.

Der Test des Mikroprogrammspeichers (ROM) und des Pipeline-Registers erfolgt mit Hilfe der Signaturanalyse [3]. An zusätzlicher Hardware ist ein Zähler (mit Null-Erkennung) am Adreßeingang des Speichers notwendig. Das Pipeline-Register kann als rückgekoppeltes Schieberegister aufgebaut und damit als Signaturregister eingesetzt werden [3]. Der Testprozessor setzt den Zähler am Testanfang, steuert den

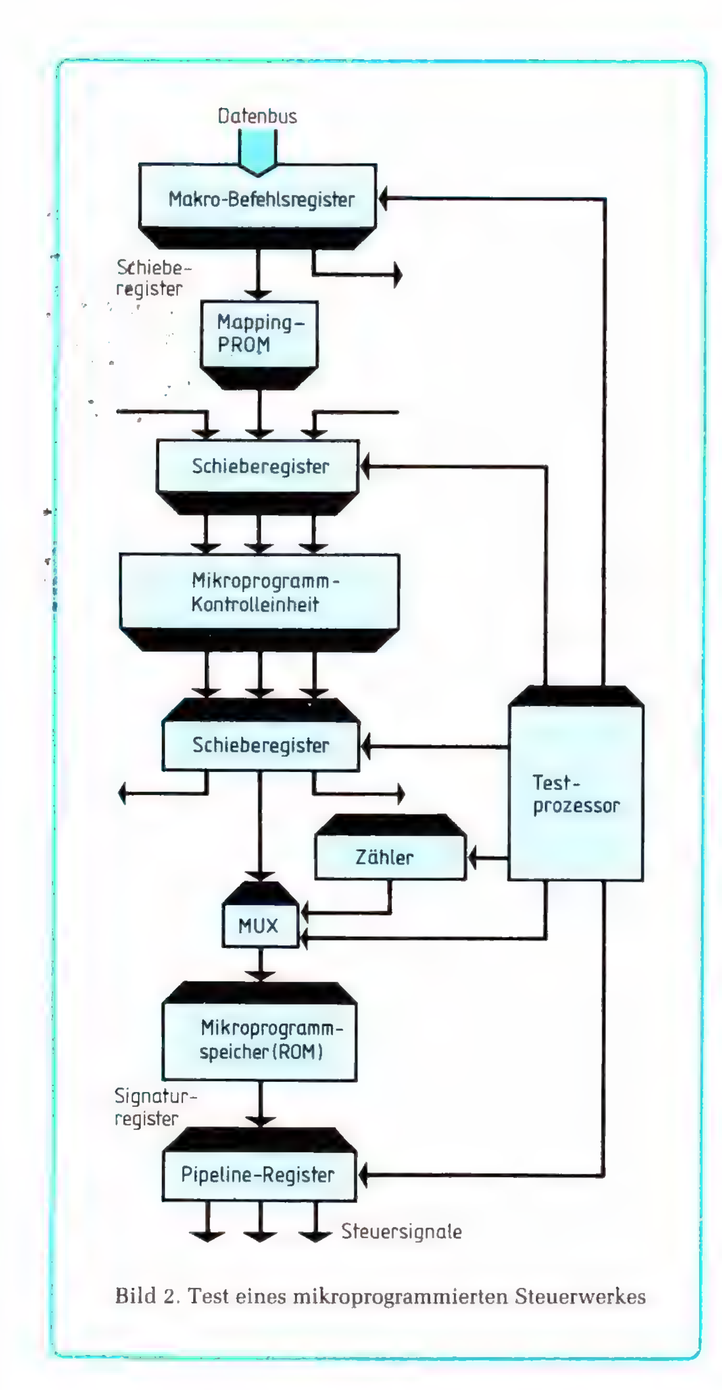


Bild 2. Test eines mikroprogrammierten Steuerwerkes

Multiplexer zwischen Zähler und Mikrokontroller, stellt das Signaturregister auf einen bestimmten Anfangswert ein und wertet das Ergebnis im Signaturregister aus. Solch ein Test überprüft sowohl den Inhalt des Speichers als auch die Adreßlogik.

Eine Mikroprogramm-Kontrolleinheit in der Komplexität eines AM 2910 hat zum Beispiel 20 Eingänge und 16 Ausgänge. Sie kann vom Testprozessor durch Anlegen von Testmustern an die Eingänge und durch Auswerten der Ausgänge getestet werden. Dazu sollten alle Ein- bzw. Ausgänge an jeweils einem Schieberegister hängen, das vom Testprozessor seriell geladen bzw. gelesen werden kann.

Testmuster und Soll-Ergebnisse können wahrscheinlich nicht auf dem Chip gehalten werden. Wegen der geringen Komplexität des Controllers (nur 16 Befehle) lassen sie sich eventuell algorithmisch vom Testprozessor erzeugen. Ansonsten müssen sie jeweils von außen über die serielle Schnittstelle, die der Testprozessor zur Verbindung mit einem Serviceprozessor hat, geladen werden.

Makro-Befehlsregister und Mapping-PROM lassen sich wiederum mit Hilfe der Signaturanalyse testen. Wegen der geringen Breite des Eingangs (8-Bit-Opcode) kann die Funktion des Zählers als Stimuli eventuell auch vom Testprozessor übernommen werden. Das Schieberegister zwischen Mapping-PROM und Mikrocontroller wird so aufgebaut, daß es als Signaturregister verwendbar ist. Bild 2 verdeutlicht noch einmal die zusätzliche Testlogik und die Verbindungen zum Testprozessor, die für das Überprüfen des Steuerwerkes notwendig sind.

Nachdem der Testkern des Funktionsprozessors durch den Testprozessor überprüft wurde, kann sich nun der Prozessor mit Hilfe von Mikro-Diagnoseprogrammen, die im ROM abgespeichert sind, selbst testen. Die Ergebnisse des Prozessor-Selbsttests können in einem Statusregister, welches als erstes Testobjekt überprüft werden muß, abgelegt werden. Der Testprozessor kann den Inhalt dieses Statusregisters nach außen weiterleiten.

4 Ergebnisse

Die Komplexität des Testprozessors ist gegenüber der des eigentlichen Funktionsprozessors relativ gering. Erste Abschätzungen ergaben einen Befehlssatz von 40...50 Befehlen, eine Wortbreite von 8 Bit, 256 Worte ROM und 16 Worte RAM. Für die Kommunikationsschnittstelle zum Serviceprozessor sind ca. 10...15 zusätzliche Anschlüsse am Prozessorchip erforderlich.

Literatur

[1] *Ebel, B.*: Mikroprozessor Selbsttest. Elektronische Rechenanlagen 1978, H. 4, S. 186...194.

[2] *Nilsson, S.-A.*: Selbsttestverfahren von Mikrorechnern. VDI-Berichte, Nr. 328 (1978), S. 77...86.

[3] *Könemann, Mucha, Zwiehoff*: Built-in Test for Complex Digital Integrated Circuits. ESSCIRC 79, Southampton, S. 89...90.

Dr.-Ing. Werner Hoffmann ist in Erlangen geboren. Er studierte dort Mathematik und promovierte 1978 in Informatik. 1979 trat er bei der Siemens AG in Erlangen ein. Zunächst beschäftigte er sich mit dem Testen von VLSI-Schaltungen, seit kurzem ist er im Bereich Leistungsbewertung tätig.
Hobbys: Fotografieren, Garten
Diensttelefon: (0 91 31) 7 69 52



Dr. Reinhard Männer, Dipl.-Phys. Bruno Deluigi

Busvergabe durch dezentralen Arbiter

Ein wichtiger Gesichtspunkt beim Entwurf von Mikroprozessorsystemen ist die Kapazität des Bussystems. Bei komplexeren, stark gekoppelten Systemen, die aus einer großen Zahl von Einzelprozessoren bestehen, reicht ein einzelner Bus zur Bewältigung der anfallenden Datentransfers nicht aus. Daher müssen andere Lösungen angestrebt werden. Beispiel dafür ist ein nicht beschränktes Bussystem mit der hier beschriebenen fairen Busvergabe durch einen dezentralen, parallelen Arbiter.

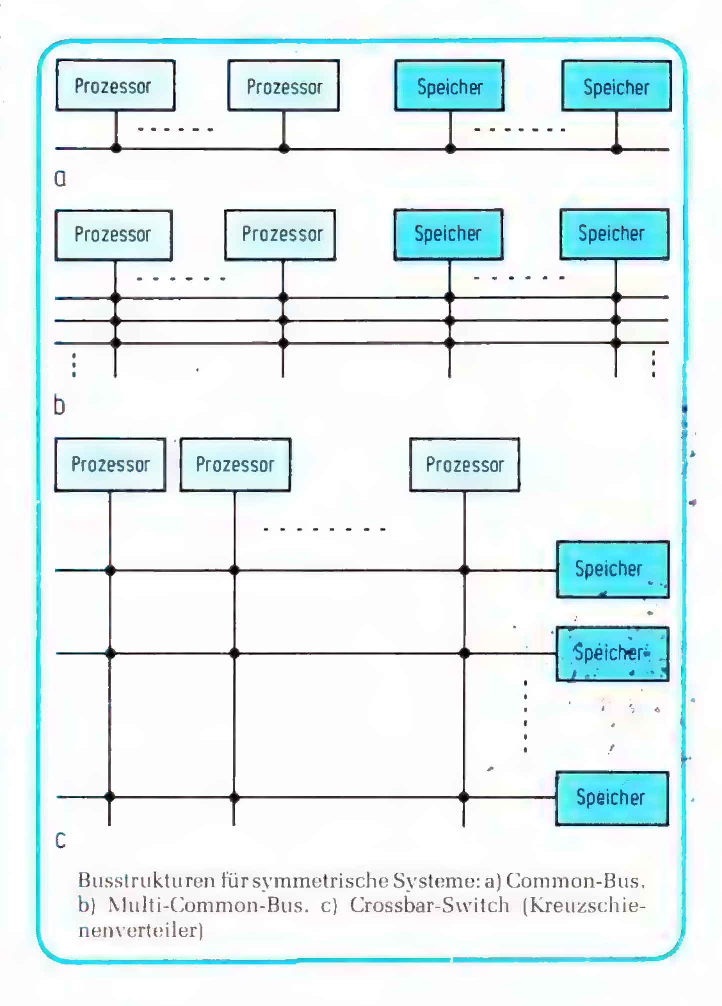
1 Konzepte

Mikroprozessoren lassen sich auf unterschiedliche Art zu einem Multiprozessorsystem zusammenfassen, beispielsweise indem verschiedene Spezialprozessoren untereinander verbunden werden, die so die Funktion eines konventionellen Rechners erfüllen. Ein anders geartetes System läßt sich durch einen vollständig symmetrischen Verbund von gleichartigen Prozessoren realisieren. Diese Architektur eignet sich vor allem, um anfallende Aufgaben auf die einzelnen Prozessoren zu verteilen und parallel zu verarbeiten. Mit solchen Systemen lassen sich große Rechenkapazitäten erre ien; dies gestattet es, etwa in der Prozeßdatenverarbeitung, aufwendige Hardware durch Software zu ersetzen. Man erreicht eine sehr hohe Anpassungsfähigkeit an verschiedenste Probleme. Wachsende Softwareanforderungen lassen sich durch eine größere Anzahl von Prozessormoduln ausgleichen. Dabei muß aber auch das Bussystem, das alle Moduln verbindet, den erhöhten Ansprüchen entsprechend erweitert werden können.

In lose gekoppelten Multiprozessorsystemen vermag meist ein einzelner Bus den Datentransfer zu bewältigen. Bei der Realisierung größerer, stark gekoppelter Systeme hingegen ist die Bereitstellung von ausreichenden Kommunikationsmöglichkeiten ein zentrales Problem. Stark gekoppelte Systeme sind durch einen intensiven Datenfluß gekennzeichnet, z. B. zwischen den einzelnen Prozessoren sowie zwischen Prozessoren und wechselseitig benutzten gemeinsamen Speichern und anderen Systemteilen.

Hier sind Bussysteme erforderlich, die eine gleichzeitige Datenübertragung auf mehreren unabhängigen Einzelbussen erlauben. Aus ökonomischen Gründen muß in der Regel jeder Einzelbus von jedem der asynchron arbeitenden Prozessoren benutzt werden können. Die Verwaltung solcher Mehrfach-Busstrukturen, d. h. die Vergabe von freien Bussen an anfordernde Prozessoren nach gewissen Kriterien, ist deshalb eine wesentliche Aufgabe innerhalb von Multiprozessor-Systemen; die Art ihrer Realisierung beeinflußt die Systemeffizienz in der Regel stark.

Symmetrische Multiprozessorsysteme bieten sich für eine dezentrale Hard- und Software-Organisation an; selbst traditionell zentrale Systemteile wie das Betriebssystem können dezentralisiert und so angelegt werden, daß ihnen die aktuelle Zahl benutzbarer Moduln, Busse usw. unbekannt ist. Fallen einzelne Komponenten aus, werden sie routinemäßig nicht mehr



Busstrukturen für symmetrische Systeme: a) Common-Bus, b) Multi-Common-Bus, c) Crossbar-Switch (Kreuzschienenverteiler)

verwendet; dies beeinträchtigt die Systemeffizienz wenig.

Diesen beiden Anforderungen, nämlich Flexibilität und Fehlertoleranz, muß auch das Bussystem genügen.

2 Bussysteme

Die am häufigsten vorgeschlagenen und zum Teil realisierten Busstrukturen für symmetrische Systeme sind der übliche Common-Bus, der Multi-Common-Bus und der Crossbar-Switch (*Bild*).

Der Common-Bus ist ein einzelner Bus, der keine parallelen Datentransfers gestattet; er erfüllt beide oben genannten Bedingungen nicht.

Nach dem Prinzip des Kreuzschienenverteilers (*Crossbar-Switch*) lassen sich aktive und passive Moduln wie z. B. Prozessoren und Speicher miteinander verbinden. Die Kapazität eines solchen Systems ist sehr groß, da eine Verbindung zu einem freien Modul immer möglich ist, unabhängig von den parallel dazu stattfindenden Kommunikationen. Es ist jedoch nur mit großem Aufwand zu erweitern, und zudem ist seine Fehlertoleranz gering, da jeweils zwei Moduln nur über einen einzigen Weg miteinander kommunizieren können.

Beim Multi-Common-Bus besitzen alle Moduln einen eigenen Bus; diese einzelnen Busse werden durch eine beliebige Anzahl von parallelen Bussen untereinander verbunden. Ein Datentransfer führt also über die zwei Modulbusse und einen der verbindenden Datenbusse.

Verbindungen sind zwischen allen Moduln in gleicher Weise möglich; es existieren jedoch mehrere Wege über die parallelen Datenbusse. Defekte Busschalter können darum umgangen werden; dies führt zu einer hohen Fehlertoleranz.

Bei geeigneter Architektur ist die Zahl der Datenbusse nicht beschränkt. Es lassen sich beliebige Buskapazitäten verwirklichen.

3 Bus-Arbitration

In einem Multiprozessorsystem mit einem Multi-Common-Bus wird man die Anzahl der Datenbusse nur so groß wählen, daß die Leistungsfähigkeit des Rechnersystems im Mittel nicht durch die Buskapazität beschränkt wird. Ein freier Bus kann deshalb von einem Prozessor nicht direkt, sondern erst nach Anforderung und Zuteilung belegt werden. In jedem gebräuchlichen Rechnersystem wird diese Aufgabe von einem Bus-Arbiter übernommen, der anhand geeigneter Kriterien von allen Anforderungen eine auswählt. Bei einem Multi-Common-Bus ist zusätzlich eine Auswahl eines freien Busses notwendig.

Ein zentraler Bus-Arbiter, der wie in Einzelbus-Systemen die Anforderungen seriell bearbeitet, läßt sich auch für den Multi-Common-Bus ohne Schwierigkeiten realisieren: Jeder Prozessor meldet etwa über eine eigene Leitung einer Busvergabelogik die Priorität seiner Anforderung. Von allen Prozessoren mit der höchsten Priorität wählt der Arbiter einen beliebigen aus und teilt diesem den ersten freien Bus zu. Nun wird mit dem nächsten freien Bus genauso verfahren. Die Nachteile dieses Verfahrens liegen auf der Hand:

- Das System kann nur umständlich erweitert werden, da jeder Teilnehmer separat an den Arbiter angeschlossen wird.
- Das System ist fehleranfällig: ein Defekt im Arbiter hat fatale Folgen.
- Die Buskapazität ist stark begrenzt: nach einer bestimmten Anzahl von Buszuteilungen wird ein vergebener Bus wieder frei und muß erneut verwaltet werden. Damit ist die Grenze erreicht; mehr Busse können mit diesem Prinzip nicht sinnvoll genutzt werden.

Bei den üblichen Busvergabe-Methoden gibt es zudem keine völlig gleichberechtigten Moduln. Bei mehreren Anforderungen mit der gleichen Priorität wird diejenige bevorzugt, welche am nächsten beim Arbiter liegt. Dies kann dazu führen, daß ein Modul mit einer hohen Anforderungsrate alle Komponenten blockiert, die weiter vom Arbiter entfernt sind. Ein solches Busvergabe-Verfahren erfüllt nicht die Ansprüche, die an den Bus eines großen Multiprozessorsystems gestellt werden; dies gilt ebenfalls für die anderen, bisher realisierten Busvergabe-Prinzipien.

Das hier beschriebene Arbitrierungs-Verfahren weist folgende Eigenschaften auf:

- Die gesamte Vergabe-Logik ist vollständig dezentral aufgebaut und damit fehlertolerant. Der Ausfall einzelner Komponenten führt dazu, daß entweder einzelne Kreuzungspunkte oder schlimmstenfalls einzelne Moduln bzw. Datenbusse unbenutzbar werden. Dies führt nur zu einer geringfügigen Verlangsamung des Systems.
- Die Bus-Arbitrierung ist so strukturiert, daß sowohl die Zahl der angeschlossenen Moduln, als auch die Zahl der verwendeten Datenbusse nicht beschränkt ist; die Busvergabe-Logik ist in beiden Richtungen kaskadierbar und damit sehr flexibel.
- Busanforderungen werden nach Prioritäten bearbeitet. Dies ermöglicht kurze Zugriffszeiten bei Echtzeitanwendungen bei gleichzeitiger ökonomischer Dimensionierung des Bussystems.
- Busanforderungen gleicher Prioritäten werden fair, d. h. ohne jede Bevorzugung bearbeitet.
- Die Busvergabe erfolgt sehr schnell: Die Zeit zwischen einer Busanforderung hoher Priorität und der Zuteilung eines freien Busses liegt typisch zwischen 0 und 100 ns; innerhalb von 200 ns können beliebig viele Busse vergeben werden.

4 Realisierung einer dezentralen, parallelen Bus-Arbitration

Die oben diskutierten Eigenschaften lassen sich mit folgendem Prinzip der Bus-Arbitrierung erzielen: Auf jedem freien Datenbus zirkuliert völlig unabhängig ein Zuteilungssignal; es startet am Busanfang und passiert sämtliche Kreuzungspunkte des Datenbusses mit den Modulbussen. Wenn das Signal am Ende angekommen ist, wird es erneut gestartet.

Falls ein Modul einen Datenbus benutzen will, erzeugt es auf dem Modulbus, und damit an allen Kreuzungspunkten, ein Anforderungssignal. Dieses Signal hält die nächste Flanke des Zuteilungssignals auf einem beliebigen freien Datenbus auf; dieser betreffende

Bus kann nun nicht mehr anderweitig vergeben werden. Unmittelbar nach Eintreffen der Flanke wird das Anforderungssignal zurückgenommen, so daß alle folgenden Zuteilungssignale von anderen freien Bussen den entsprechenden Modulbus passieren können.

Folgen jedoch zufällig einige Zuteilungssignale sehr dicht aufeinander, so können mehrere am Modulbus festgehalten werden. Von den entsprechenden Datenbussen wird einer durch die modulinterne Modulbus-Arbitrierung ausgewählt; die übrigen werden sofort wieder freigegeben.

Nach beendetem Datentransfer läuft das Zuteilungssignal weiter über die restlichen Kreuzungspunkte und zirkuliert von neuem, bis es durch eine Zuteilung erneut festgehalten wird.

Die für jeden Kreuzungspunkt erforderliche Logik ist sehr einfach und so ausgelegt, daß sie sich beliebig kaskadieren läßt; lediglich der Anfang eines Busses besitzt einen Kurzschlußstecker.

Dadurch, daß das Zuteilungssignal immer dort weiterläuft, wo es angehalten wurde, ist gewährleistet, daß alle Moduln fair behandelt werden, d. h. alle haben dieselbe Wahrscheinlichkeit, einen Bus zu bekommen. Zudem ist ausgeschlossen, daß ein Prozessor mit einer hohen Anforderungsrate alle anderen blockiert.

Von mehreren freien Bussen bekommt jeder Modul immer denjenigen zugewiesen, der am schnellsten verfügbar ist. Da die einzelnen Busse unabhängig voneinander und gleichzeitig vergeben werden, lassen sich beliebig viele Busse zur Erhöhung des Datenflusses einsetzen; damit kann das Multi-Common-Bussystem an die Zahl der eingesetzten Moduln angepaßt werden.

In einem Multi-Tasking-System läßt sich ein Bussystem effizienter nutzen, wenn die Busanforderungen nach den Prioritäten der anfordernden Tasks bearbeitet werden.

Die oben beschriebene Bus-Arbitrierung läßt sich sehr leicht mit Prioritäten versehen:
Zusätzlich zu den einzelnen Datenbussen werden alle Modulbusse durch einen Prioritätenbus verbunden, auf dem für jede Anforderungspriorität eine Leitung reserviert ist. Fordert ein Modul einen Datenbus an, so legt es zunächst nur ein Signal auf diejenige Leitung des Prioritätsbusses, die seiner Anforderungspriorität entspricht. Gleiche Signale verschiedener Moduln werden mit der ODER-Funktion verknüpft.

Jeder Modul kann nun für sich entscheiden, ob seine Anforderung der augenblicklich höchsten Priorität entspricht. Nur in diesem Fall wird das oben erwähnte Anforderungssignal auf den Modulbus und zu den einzelnen Knotenpunkten weitergegeben. Die zirkulierenden Zuteilungsflanken werden dann zunächst alle Anforderungen der augenblicklich höchsten Priorität bedienen, worauf diese ihre Priorität vom Prioritätenbus entfernen. Darauf können auch für weniger wichtige Tasks Anforderungssignale erzeugt werden.

5 Implementierung

Ein Multi-Common-Bus mit der beschriebenen Bus-Arbitierung befindet sich zur Zeit in Heidelberg in der Erprobungsphase. Dieses Bussystem soll mit 5...15 Einzelbussen innerhalb eines Multi-Mikroprozessorsystems verwendet werden, anwendungsabhängig sind bis zu 300 Prozessoren vorgesehen.

Literatur

1 Färber, G.: Ein dezentralisierter fairer Bus-Arbiter. ELEKTRONIK. 1980, H. 8, S. 65...68.



Dipl.-Phys. Dr. R. Männer wurde in Weiherhammer/Opf. geboren. Er studierte in Munchen Experimentalphysik. Nach seinem Diplom wechselte er 1975 an das Physikalische Institut der Universität Heidelberg, wo er mit der Entwicklung eines Realtime-Multitasking-Paging-Betriebssystems für die PDP11/45 promovierte. Seit 1979 leitet er die Entwicklung eines Multi-Mikroprozessorsystems auf der Basis des MC68000. Hobbys: Klettern, Bergsteigen, Musik
Diensttelefon: (0 62 21) 5 69-3 63



Dipl.-Phys. Bruno Deluigi ist Bürger von Sala Capriasca/Tessin. Nach einer Ausbildung in Elektrotechnik an der Ingenieurschule Winterthur begann er ein Physikstudium an der ETH in Zürich, das er in Heidelberg 1978 mit dem Diplom beendete. Seit 1979 arbeitet er mit an der Entwicklung eines Multiprozessorsystems.
Diensttelefon: (0 62 21) 48 45 76

Peter L. Andersen

Verteiltes System mit Einplatinen-Computern

Problemlösungen hoher Komplexität werden in zunehmendem Maße in der Form verteilter Systeme realisiert. Vorteile, die sich dabei ergeben, sind z. B. vereinfachter Aufbau und dadurch geringere Kosten sowie ein transparentes, leicht erweiterbares Konzept. Allerdings ist es wichtig, ein brauchbares Kommunikationsschema zu verwenden, das für die Verbindung der einzelnen Systembestandteile sorgt. Hier wird an einem Beispiel gezeigt, wie ein verteiltes System mit Einplatinen-Computern aufgebaut ist, bei dem standardmäßige Hard- und Softwarebestandteile verwendet werden.

Verteilte Systeme lassen sich grundsätzlich in zwei Gruppen aufteilen, nämlich Konfigurationen, bei denen die Funktionsgruppen so angeordnet sind, daß diese über einen gemeinsamen parallelen Datenbus kommunizieren können, und andere, bei denen die Funktionsgruppen örtlich so weit voneinander getrennt sind, daß sie über eine serielle Schnittstelle miteinander verbunden werden müssen.

1 Anwendungsbeispiel

Das Computersystem in diesem Beispiel dient zur Überwachung und Sicherung eines Komplexes, der aus mehreren Gebäuden besteht (*Bild* 1). Das System überwacht die drei bereits existierenden Gebäude, das geplante vierte soll nach Fertigstellung ebenfalls angeschlossen werden, in jedem Gebäude ist eine Kontrollstation vorgesehen, außerdem eine Zentrale, mit dem sich der Gesamtkomplex übersehen läßt. Für Anwendungen dieser Art ist ein Verteilen der Systemfunktion über den Gebäudekomplex mit zwei Vorteilen verbunden: Die Installationskosten verringern sich und die Zuverlässigkeit des Systems wird erhöht.

2 Hardware-Implementierung

2.1 Sternförmiges Netzwerk

In einem sternförmigen Netzwerk kann eigentlich jedes Übertragungsverfahren verwendet werden. Weil immer nur eine Einheit mit einer anderen kommuniziert, spielt die Kommunikationsform eine untergeordnete Rolle. In diesem Beispiel wird die RS232C-Datenübertragung angewendet.

Herz der Stern-Konfiguration ist eine Master-Einheit. In ihr müssen die Mittel vorgesehen werden, die die Kommunikation mit den Untereinheiten (Slaves) ausführen. Eine kostengünstige Lösung ergibt sich, wenn die Kommunikationstreiber auf einer gemeinsamen Platine untergebracht sind. In dem hier beschriebenen Anwendungsbeispiel sind vier Slaves notwendig, die vom Master gesteuert werden. Diese Funktion wird von der intelligenten Steuereinheit iSBC 544 ausgeführt. Diese Platine arbeitet mit einer Baudrate, die softwaremäßig einstellbar ist. Die Intelligenzfunktionen entlasten den Hostrechner von den Kommunikationsaktivitäten. Wenn außer den vier Untereinheiten weitere Slaves benötigt werden, kann das System durch einfaches Hinzuführen einer solchen Kommunikationseinheit erweitert werden.

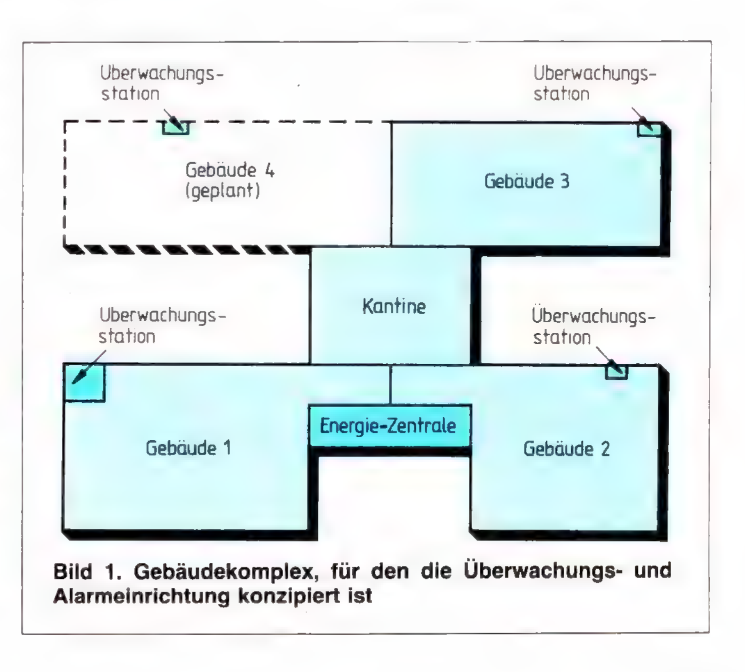


Bild 1. Gebäudekomplex, für den die Überwachungs- und Alarmeinrichtung konzipiert ist

Als Slave-Prozessoren bieten sich die Einplatinen-Computer iSBC 80/108 an, die preiswert sind und ausreichende Verarbeitungskapazität bieten. Die Platinen besitzen sowohl ein RS232C- als auch ein 20-mA-Stromschleifen-Interface und lassen sich sowohl in der RS232- als auch in EIA-Betriebsart verwenden. Der EIA-Betrieb unterscheidet sich vom RS232C-Standard nur dadurch, daß eine größere Übertragungsdistanz, allerdings bei geringerer Baudrate, möglich ist. In dem Anwendungsbeispiel sind beide Betriebsarten möglich.

Die auf der Kommunikationsplatine vorzusehenden Verbindungen zur Einstellung der Betriebsart werden am Stecker J1 vorgenommen (*Bild 2*). Zu beachten ist, daß die Signale „Ready to Send" und „Clear to Send" miteinander verbunden werden müssen, um das korrekte Arbeiten der USART ohne die Steuerung durch ein Modem zu gewährleisten. Die Sende- und Empfangssignale werden außerdem miteinander vertauscht.

Wenn ein RS232C-Signal über eine längere Distanz übertragen wird, muß das aus Gründen der Zuverlässigkeit mit niedriger Geschwindigkeit geschehen. Im hier beschriebenen Beispiel ist eine Baudrate von 1200 ausreichend, denn das Auslagern von Intelligenz in die Untereinheiten minimiert die Datenmengen, die übertragen werden müssen.

Während der Entwicklungsphase der Hardware muß auch das Kommunikationsschema für die Übertragung zwischen Master und Slave festgelegt werden. Dieses Problem kann sehr einfach gelöst werden, denn die Kommunikationsplatine iSBC 544 verfügt über Master/Slave-Protokoll-Funktionen. Diese werden von drei Hardwaremerkmalen unterstützt. Das erste ist der zweiseitige Speicher Dual-Port-Memory auf der Platine. Hier können Daten und Befehle für die Kommunikation zwischen Slave und Host-Prozessor gespeichert werden. Die zweite Einrichtung erzeugt immer dann, wenn der Hostcomputer in den ersten Platz des Dual-Port-RAMs einschreibt, einen Flag-Interrupt. Der Interrupt wird zurückgesetzt, wenn der Speicherinhalt vom Slave-Prozessor gelesen wurde. Drittens erzeugt die Ausführung der SOD-Instruktion des 8085A-2 einen Multibus-Interrupt, der den Hostcomputer davon unterrichtet, daß der Slave bedient werden muß. Mit diesem Interrupt kann eine Service-Routine zur Bedienung des Slave-Boards aktiviert werden. Eine ausführliche Beschreibung der Merkmale des iSBC 544 findet man in [1].

Bild 2. Verdrahtung des Platinensteckers zur Einstellung der gewünschten Betriebsart

Bild 3. Verdrahtung der Leitungsanschlüsse: a) Master-Platine, b) Slave-Platine

2.2 Ringstruktur

Im Gegensatz zur Sternkonfiguration unterliegt die Hardware der ringförmigen Kommunikationsleitung gewissen Beschränkungen. Es muß ein Übertragungsschema verwendet werden, das es jeder Untereinheit erlaubt, nacheinander die Kontrolle über das Kommunikationsnetz zu erlangen, um eine Nachricht zu übertragen.

In einem Ringnetz ist sowohl Halb- als auch Voll-Duplex-Betrieb möglich. Während der Hardwareaufwand bei einer Halb-Duplex-Übertragung geringer ist, muß das Kommunikationsprotokoll in diesem Fall etwas enger ausgelegt werden, weil die Übertragung immer nur in eine Richtung möglich ist. Aus praktischen Gründen ist keine Priorität für Master oder Slaves vorgesehen. Alle Teilnehmer empfangen die Signale,

egal wer sie absendet. Die Software muß so konzipiert sein, daß immer nur eine Untereinheit kommunizieren kann. Ein Beispiel für ein Halb-Duplex-Netzwerk ist das Ethernet.

Software für Voll-Duplex-Betrieb ist unkomplizierter. In diesem Fall wird vorausgesetzt, daß im System nur ein Master vorgesehen ist, der die Ausgangsleitungen treibt. Eine beliebige Zahl von Slaves treibt die Eingangsleitungen. Die Kommunikation wird auf Anforderung des Masters ausgeführt.

Für den Betrieb im Netzwerk müssen einige auf der Platine befindliche Optionen aktiviert werden. Die erste Option ist die Konfigurierung der Ausgangstreiber als Tri-State-Treiber. Dadurch ist es möglich, daß nur die Platine, die die Steuerung der Kommunikationssignale übernimmt, Daten auf den Ausgangsleitungen absetzen kann. Die Konfiguration von Master und Slave unterscheiden sich (*Bild 3*). Die Ausgangssignal-Leitungen werden durch das DTR-Signal (Data Terminal Ready) der USART gesteuert.

Bild 4 zeigt den Verdrahtungsaufwand, der für ein Voll-Duplex-Ring-Netzwerk mit einem Master und vier Slaves erforderlich ist. Für den Abschluß der Kommunikationsleitungen werden Widerstände vorgesehen. Eine genaue Bestimmung der Widerstandswerte ist in [2] beschrieben. Für den RS422-Sektor des Boards sind Widerstände zur Erzeugung der Vorspannung vorzusehen, die sich überschlagsmäßig folgendermaßen bestimmen lassen:

$R_{B1} = 4500/(18{,}6 - 1{,}6 \times N)$

$R_{B2} = 2500/(18{,}6 - 1{,}6 \times N)$

(N ist die Anzahl der Slaves).

In dem hier beschriebenen Beispiel beträgt der Wert von R_{B1} 370 Ω, R_{B2} hat 205 Ω.

Auf einer verdrillten Leitung kann mit einer Baudrate von 19,2 kBd nach dem RS422-Standard eine Entfernung von etwa 1,2 km überbrückt werden. Das ist für die hier beschriebene Anwendung ausreichend, da die größte Entfernung bei etwas mehr als 500 m liegt.

3 Software-Implementierung

Nach der Phase der Hardwaredefinition müssen zunächst die Protokolle definiert werden, nach denen die Datenübertragung und die gegenseitige Steuerung („Handshaking") ausgeführt wird. Das Handshaking ist die Hintergrundkommunikation, mit der die Synchronisierung und Integrität der Kommunikationseinrichtung gewährleistet wird.

3.1 Beziehung zwischen Master und Slave

Um einen geordneten Fluß von Daten zwischen den Stationen, die am Netz angeschlossen sind, zu gewährleisten, muß die Software Prioritäten zuweisen, damit zu einem Zeitpunkt immer nur ein Teilnehmer sendet. Die einfachste Möglichkeit dazu ist eine Master-Slave-Hierarchie. Dabei wird ein Teilnehmer als Master definiert und die anderen als Slaves. Jedem Slave wird eine Zeit zugewiesen, in der er anderen Teilnehmern Daten übermitteln kann. Der Master steuert die Priorisierung der Slaves. Die Komplexität des Systems kann reduziert werden, wenn ein Voll-Duplex-Übertragungsverfahren gewählt wird. In dieser Betriebsart empfangen alle Slaves die Signale des Masters und übertragen ihre Nachrichten über einen gemeinsamen seriellen Kanal zum Master. Ein Nachteil dieser Technik ist die Tatsache, daß Übertragungen zwischen den Slaves direkt nicht möglich sind.

Bild 5 zeigt, wie ein Master die Zeitfenster festlegt, während deren ein Slave die Kontrolle über den Ausgabekanal zur Übertragung von Nachrichten zum Master übernehmen kann. Jedem Slave ist eine Adresse zur Identifizierung zugeordnet. Jedes Datenpaket enthält eine Adresse. Wenn ein Slave seine Adresse im Paket, das vom Master ausgesendet wurde, erkennt, steht ihm ein festes Zeitfenster zur Verfügung, in dem eine Nachricht zum Master zurückgesendet werden kann.

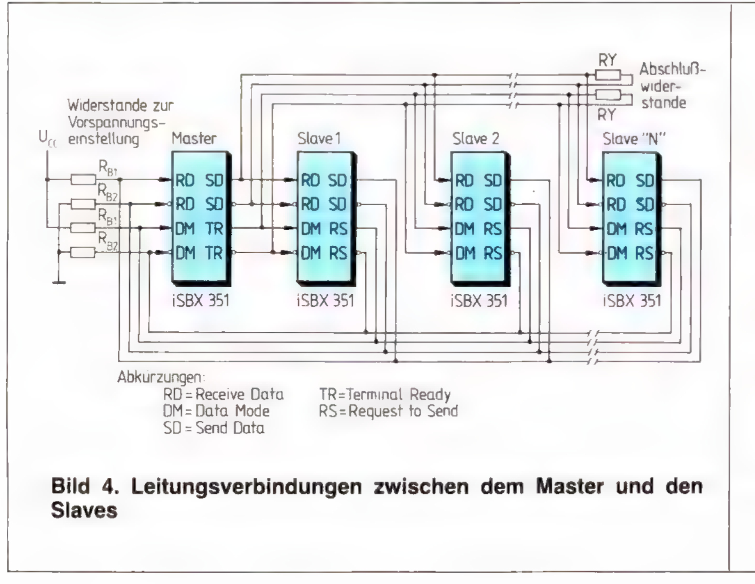


Bild 4. Leitungsverbindungen zwischen dem Master und den Slaves

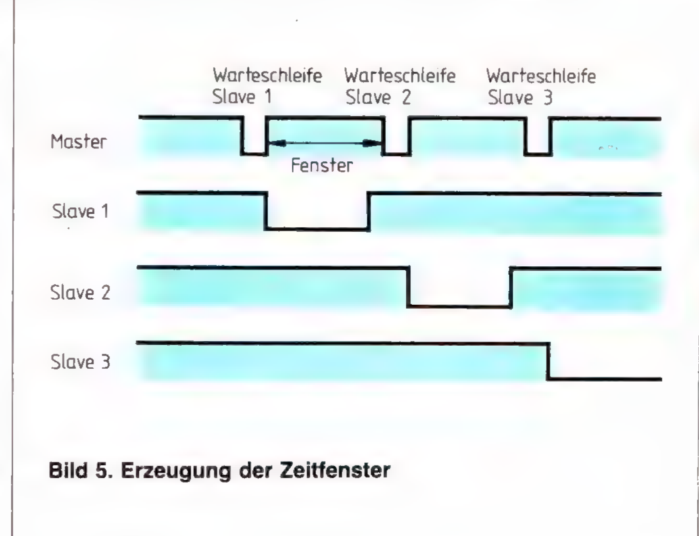


Bild 5. Erzeugung der Zeitfenster

3.2 *Datenpaket-Formate*

Die Kommunikation zwischen jedem Slave und dem Master besteht aus der Übertragung von Datenpaketen. Jedes Paket enthält die Handshaking-Information (z. B. Checksumme und Anzahl der Nachrichten) sowie die Informationen, die zwischen den Stationen des Netzes ausgetauscht werden sollen. Die Handshaking-Abschnitte des Übertragungssignals werden auch dazu benutzt, die Integrität der Übertragung in bezug auf externe Einflüsse, z. B. Störungen, zu überwachen.

Debugging und Systemwartung werden minimiert, wenn eine Einrichtung vorgesehen wird, mit der die Kommunikationssignale leicht überwacht und interpretiert werden können. In dem hier beschriebenen Beispiel wird ein ASCII-kompatibles Kommunikationsformat verwendet. Dadurch ist es möglich, mit einem CRT-Terminal, das an die Signalleitung angeschlossen wird, die Kommunikation zu überwachen.

Jedes Nachrichtenpaket beginnt und endet mit einem bestimmten Zeichen. Dieses Zeichen kann nicht innerhalb der Nachricht verwendet werden. Zu Beginn eines Nachrichtenpaketes wird „Line Feed“ und zum Ende „Carriage Return“ gesendet. Durch diese Zuweisung kann ein Nachrichtenpaket leicht interpretiert werden.

Die Zusammensetzung eines Datenpaketes ist in *Bild* 6 dargestellt. Ein großer Teil der Nachricht ist zur Sicherstellung der Systemintegrität vorgesehen. Die Notwendigkeit einer Adresse wurde bereits erwähnt. Zwei Zeichen genügen, um 256 unterschiedliche Adressen (00...FF H) zu erzeugen. Dem Master wird die Adresse 0 zugewiesen, die restlichen 255 Adressen stehen für Slaves zur Verfügung.

Das nächste Feld beschreibt die Art der Nachricht, die gesendet wird, und die Aktion, die vom Empfänger ausgeführt werden muß. Eine Analyse von Erfordernissen der Systemkommunikation zeigt, daß fünf Nachrichtentypen ausreichend sind. Mit einem einzigen Byte in diesem Feld können insgesamt 16 Typen unterschieden werden. Damit bleiben elf für spätere Erweiterungen frei. In diesem Beispiel werden folgende Typen verwendet:

– *Typ 0:* Dies ist eine Aufforderung des Masters an den Slave für einen Reset. Der adressierte Slave generiert einen Software-Reset. Dieses Kommando wird nur benutzt, wenn die Kommunikation durch andere Mittel nicht aufgebaut werden kann.
– *Typ 1:* Dieses Kommando wird zur Synchronisierung von Master und Slave benutzt. Der Message-Zähler wird auf 0 zurückgesetzt. Dieses Kommando kann nur vom Master ausgegeben werden.
– *Typ 2:* Die Nachricht vom Typ 2 ist als diejenige definiert, die die Information enthält, die zwischen den einzelnen Stationen des Kommunikationsnetzes weitergeleitet wird. Das exakte Format dieser Daten ist woanders festgelegt. Dies ist die einzige Nachricht, die nicht die Handshaking-Natur aufweist.
– *Typ 3:* Diese Message zeigt an, daß der Teilnehmer erreicht wurde und bereit ist, zu reagieren. Wenn eine normale Kommunikation aufgebaut wurde und keine Nachricht ausgesendet wird, wird diese Message zwischen Master und Slave dauernd gesendet.
– *Typ 5:* Diese Nachricht wird vom Slave benutzt, um auf die Aufforderung des Masters zur Synchronisierung zu antworten. Die Quittierung dieses Signals zeigt an, daß der Slave alle notwendigen Operationen ausgeführt hat und die Message-Zähler bereit sind, neue Nachrichten zu empfangen.

Das nächste Feld wird dazu benutzt, die Anzahl der Messages anzugeben, die Daten enthalten und von jedem Teilnehmer empfangen oder gesendet werden müssen. Das erste Zeichen dient zur Anzeige der Zahl der Messages, die vom Master ausgesendet und vom Slave empfangen wurden. Das zweite Zeichen zeigt die Zahl der Messages, die der Master ausgesendet hat. Das interne Softwareprotokoll benutzt diese Felder, um festzustellen, ob eine Daten-Message vom adressierten Slave korrekt empfangen wurde. Wenn die Antwort-Message nicht die richtigen Zahlen enthält, kann die Message erneut gesendet werden.

Das Datenfeld enthält die Informationen, die an den empfangenden Teilnehmer gesendet werden. Der genaue Inhalt wird in Zusammenhang mit Erweiterungen des Betriebssystems erläutert.

Zur Gewährleistung der Integrität der übertragenen Nachricht wird die Prüfsumme hinzugefügt. Dieses Feld enthält eine 8-Bit-Summe (256) aller übertragenen ASCII-Zeichen, beginnend mit „Line Feed“ bis zum letzten Zeichen des Datenfeldes. Es wird zum Vergleich der berechneten Prüfsumme der empfangenen Zeichen mit dem vom sendenden Teilnehmer ausgegebenen Zeichen verwendet. Wenn die zwei Zahlen übereinstimmen, wird die Nachricht als gültig erkannt. Das letzte Feld ist ein Zeichen, das das Ende der Nachricht kennzeichnet. Dazu wird „Carriage Return“ benutzt.

Startzeichen (Line Feed)	Zielstation (0...255)	Kommando (0...F)	Nachrichten-Zähler (0..FF)	Daten	Prüfsumme	Ende-Zeichen

durch RMX festgelegt					Erweiterung Exchange-Name (6 Zeichen)	Task-Massage-Daten
Link	Length	Type	Response Exchange	Home Exchange		
X		U/S	X	X		

Bild 6. Format des Kommunikationspaketes

3.3 *Konzepte für Multitasking-Nachrichten*

Die Entwicklung von Systemen mit verteilten Prozessoren setzt auch eine Multitasking-Umgebung voraus. Die Möglichkeit, ein Anwendungsproblem in einzelnen Tasks aufzuteilen, führt zu erheblichen Vereinfachun-

gen des Entwicklungs- und Implementierungsprozesses. In der Praxis repräsentieren allerdings nur wenige Tasks völlig isolierte Funktionen. Ein gewisser Grad von Kommunikation zwischen den Tasks ist erforderlich. Das reicht von einfacher Synchronisation bis zum Datentransfer. Dieser Prozeß wird durch ein Echtzeit-Betriebssystem vereinfacht. Der Kern des Betriebssystems iRMX 80 ist ein gutes Beispiel dafür. Es verfügt über einen kleinen Überbau, benötigt geringe Speicherkapazität und arbeitet in Zusammenhang mit fast allen 8-Bit-Platinencomputern von Intel.

Eine Message des iRMX 80 kann man mit einem Brief vergleichen, der jemandem zugeschickt wird. Dieser wird in einem Postamt abgegeben (Exchange) und wird dann dem Empfänger übergeben. Jeder Bereich der iRMX-Implementierung kann auf diese Weise benützt werden. Wenn eine serielle Kommunikationsverbindung zwischen den Prozessoren benutzt wird, entspricht das dem Versenden eines Telegramms.

Es gibt unterschiedliche Teile einer Message. Jeder Teil hat eine bestimmte Funktion und Format innerhalb der Nachrichtenstruktur. Bevor eine Nachricht erzeugt wird, muß ein Medium in Form eines RAM-Blocks bereitgestellt werden, das diese Nachricht enthält. Um immer Speicherplatz zur Verfügung stellen zu können, besitzt das Betriebssystem eine Einrichtung zur Speicherverwaltung. Deren Aufgabe ist die Wiederverwendung von Blocks, die inzwischen nicht mehr benötigt werden, zur Erzeugung weiterer Messages. Dieser Vorgang wird jedesmal ausgeführt, wenn die sendende Task die Erzeugung einer Nachricht beginnt. Wenn das Kommunikationsprogramm erfolgreich eine Nachricht zum empfangenden Teilnehmer übermittelt hat, wird der Speicherblock an den Pool der freien Speicher zurückgegeben. Wenn der Empfänger eine Nachricht erhält, muß dieser wiederum einen Block von der Speicherverwaltung erhalten, um die Information zu speichern, bis sie von der Ziel-Task verarbeitet werden kann. Diese Task gibt den Block an den Pool zurück, wenn die Aktivitäten ausgeführt worden sind.

Wie bei jeder anderen Art von Nachricht muß natürlich auch ein Empfänger existieren. Weil der Kern des iRMX 80 für die Verwendung in Zusammenhang mit mehreren Tasks auf dem gleichen Prozessor konzipiert ist, ist der Zieladreßmechanismus nicht im eigentlichen Message-Header enthalten. Aus diesem Grund muß das Ziel innerhalb des Aufrufes der iRMX-Prozedur in der Parameterliste spezifiziert werden. Wenn die Zieladresse nicht bekannt ist – ein Fall, der beim Entwurf von Multiprozessorsystemen regelmäßig auftritt –, ist es notwendig, eine Erweiterung des Betriebssystemes vorzusehen, mit der es möglich ist, jeder Message einen Zielnamen zuzuweisen. Die Erzeugung eines „Target-Exchange-Namens" ist allerdings nicht sehr kompliziert.

3.4 Erweiterung des Betriebssystemes

Der Kern des Betriebssystems iRMX 80 benutzt einen kleinen RAM-Block, um Daten zu speichern, die Eigenschaften und Zustand jeder Exchange betreffen. Dieser Bereich umfaßt in der Regel 10 Byte.

Wenn der Kern des Betriebssystemes nicht neu geschrieben werden soll, kann das Format des Exchange-Descriptors nicht geändert werden und das Identifikationsfeld muß in Form zusätzlicher Bytes diesem Speicherblock hinzugefügt werden. Zwei Merkmale des iRMX 80 machen eine direkte Implementation dieses Konzeptes allerdings unmöglich. Erstens existiert ein besonderer Exchange-Descriptor bereits. Dieser ist der Interrupt-Exchange, der fünf Byte als eine Interrupt-Nachricht hinzufügt. Zweitens müssen, wenn eine zusätzliche Erweiterung vorgesehen ist, alle Exchanges als Interrupt-Exchanges erzeugt werden, damit gemeinsame Programme verwendet werden können. Während das alleine noch kein Hinderungsgrund ist, kommt hinzu, daß der interaktive Konfigurator (ICU 80) die Hinzufügung von Feldern weder zum Standard- noch zum Interrupt-Exchange-Descriptor zuläßt. Deshalb ist eine andere Methode zur Erweiterung zulässig.

Die Operation RQCXCH erzeugt eine Exchange bei einer Adresse, die durch einen Parameter der Aufrufinstruktion spezifiziert ist. Wenn die Benutzer-Software diese Exchanges aufbaut, kann der Name jeder gewünschten Exchange vorangestellt werden.

Beim Standard-Kern des iRMX 80 sind sechs Zeichen als Name für jede Task erlaubt. Eine logische Erweiterung dieses Konzeptes führt zu einem Namen aus sechs Zeichen für jede dieser Exchanges. Der Descriptor wird daher durch Hinzufügen von sechs Byte zu seinem Format erweitert. In der Praxis ist das allerdings noch nicht ausreichend, weil diese Exchanges dynamisch erzeugt werden müssen. Der Speicherbereich, der die Exchanges repräsentiert, muß nicht unbedingt aus einem kontinuierlichen Adreßbereich bestehen. Aus diesem Grund muß ein zusätzliches Speicherwort, das einen Pointer enthält, der auf das nächste Exchange-Feld zeigt, hinzugefügt werden. Der Wert Null in diesem Feld zeigt an, daß keine zusätzlichen Exchanges vorhanden sind. *Bild 7* zeigt die Struktur dieser Felder.

Die Erzeugung von Software zur Unterstützung der Erweiterungen ist relativ unkompliziert. Zwei Funktio-

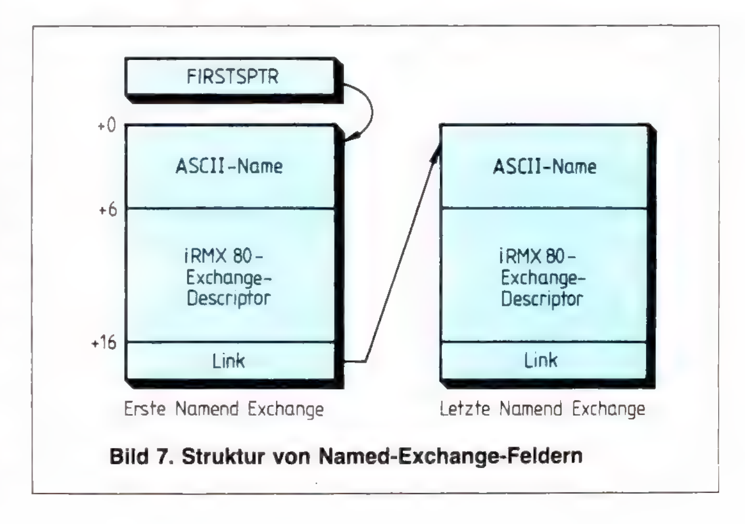


Bild 7. Struktur von Named-Exchange-Feldern

nen sind erforderlich: Eine, die die „Named Exchanges“ erzeugt und eine zweite, die die Adresse einer Exchange, die den vorgegebenen Namen entspricht, zurückgibt. Ein komplettes Listing der erforderlichen Software ist in [3] enthalten. Die Zuweisung der Speichersegmente für die Exchange-Descriptoren wird durch die Speicherverwaltung ausgeführt.

3.5 *Aufbau der seriellen Kommunikation*

Die Erstellung von Software für eine serielle Kommunikation führt zu echten Multitasking- und Multiprozessor-Fähigkeiten. Die Notwendigkeit von zwei Kommunikationspaketen für Slave- und Masterprozessor zeigt, daß zwei verschiedene Tasks erforderlich sind. Eine genaue Untersuchung der Kommunikationserfordernisse ergibt, daß das System in drei Ebenen aufgebaut werden kann (*Bild 8*). Die erste ist als Protokoll definiert und enthält den Code, der zur Implementierung des Algorithmus für Slave- und Master-Stationen erforderlich ist. Die zweite Ebene, die Verbindungsebene, enthält den Code, der für allgemeine Operationen, z. B. Message-Erzeugung und Daten-Warteschlangen-Verwaltung, benötigt wird. Die dritte Ebene besteht aus einem speziellen Interface an die physikalische Hardware für den Aufbau der Kommunikationsverbindung. Diese muß an die jeweilige Konfiguration angepaßt sein.

3.6 *Kommunikationspaket für Protokoll- und Verbindungsebene*

Für dieses Anwendungsbeispiel sind zwei Kommunikationspakete auf der Protokollebene erforderlich. Die Implementierungen für Master- und Slave-Protokoll sind vollständig in [3] abgedruckt. Das Kommunikationspaket für die Verbindungsebene sieht eine Menge von Prozeduren vor, mit denen sowohl die Protokoll- als auch physikalische Ebenen unterstützt werden können. Vollständige Listings dieser Programme sind in [3] zu finden.

Bild 8. Die Ebenen der Übertragungseinrichtung

3.7 *Kommunikationspaket der physikalischen Ebene*

Dieses Paket dient der Anpassung an die Konfigurationen des Host-Prozessors und der USART. Vier Prozeduren müssen von dieser Ebene ausgeführt werden:

- *CQ$INIT* Diese Prozedur enthält alle Operationen, die notwendig sind, um die Zeitgeber, Zähler und USARTs, die mit dem Kommunikationscode in Verbindung stehen, zu initialisieren. Außerdem definiert diese Prozedur die Interrupt-Serviceroutinen für die USARTs und gibt die entsprechenden Interruptebenen frei.
- *CQ$1START$MSG* Diese Prozedur versetzt die USART in einen Zustand, bei dem der Sender freigegeben ist. Die Ausführung der Prozedur sollte in einem Interrupt resultieren, der von der „Transmitter-Ready“-Leitung der USART generiert wird.
- *CQMIVT* Diese Interrupt-Serviceroutine hängt mit der „Receiver-Ready“-Leitung der USART zusammen. In diese Routine wird jedesmal eingesprungen, wenn ein Zeichen empfangen wurde. Für die Slave-Stationen trägt diese Prozedur die Bezeichnung CQSIVT. Der Name ist allerdings unwichtig, weil die Lage dieser Routine, bei der Initialisierung durch das CQ$INIT-Programm an den Kern des iMRX 80 weitergeleitet wird. Wenn das Zeichen für das Ende der Message (Carraiage Return) auftritt, wird eine Message an die Task der Protokollebene weitergegeben, indem mit Hilfe der iMRX-Grundoperationen RQISND eine Nachricht an die Interrupt-Exchange RQL6EX weitergereicht wird.
- *SEND$CHAR* Wie die CQMIT-Routine ist dies eine Interrupt-Handler-Prozedur. In diese Routine wird gesprungen, wenn die USART signalisiert, daß sie zur Übertragung eines Zeichens bereit ist. Dann wird ein neues Zeichen aus der Datenwarteschleife entnommen und der empfangenden Station übertragen. Wenn es sich bei diesem Zeichen um das Ende der Nachricht handelt, werden Flags gesetzt und die USART gesperrt.

Das Listing für den Treiber ist in [3] abgedruckt. In diesem Beispiel wird das serielle Multimodul-Board iSBX 351 an einen Einplatinencomputer iSBC 80/24 angeschlossen.

Literatur

[1] Using the iSBC 544 Intelligent Communications Controller. Applikations-Bericht AP-53, Intel Corp.

[2] iSBX 351 Serial MULTIMODULE Board Hardware Reference Manual. Druckschrift 9803190, Intel Corp.

[3] Using Intel Single Board Computers for Serial Processing Links. Applikations-Bericht AP-109, Intel Corp.

[4] *Pol, B.*: Betriebssysteme. ELEKTRONIK 1981, H. 2, S. 43...56.



microtronic

Computer-System

vermittelt Mikrocomputer-Technik von Grund auf – zeigt wie ein Computer funktioniert – wie man ihn programmiert

microtronic ist ein vollständiges, betriebsfertig installiertes Mikrocomputersystem. Es zeigt seinem Benutzer, nach welchen Kriterien ein Computer arbeitet. Wie er durch Befehlseingabe Daten anzeigt und speichert, zu Programmabläufen zusammenführt und auf Abruf bestimmte Funktionen ausführt. Durch Aneinanderreihen logischer und leicht erlernbarer Befehle ergeben sich unendliche Möglichkeiten, den Computer für die unterschiedlichsten Aufgaben zu programmieren.

Sie erfahren dabei alles über Bits und Bytes. Daten, Speicher und Adressen. Software und Hardware. Dual, hexadezimal, das ganze „Computer-chinesisch". Also eine ideale Einstiegsmöglichkeit in die Datenverarbeitung.

Ein echter Mikrocomputer – für unendlich viele Möglichkeiten. **DM 389.–**



microtronic ist Teil des

electronic

Experimentier-Systems

Es umfaßt vom Transistor bis zum Mikrocomputer die ganze Elektronik: Preisestand 1982. Unverbindliche Preisempfehlung

Nr.	Artikel	Preis
2059	Netzgerät	DM 33,50
2060	Compact-Studio	DM 59,90
2061	Ergänzungspackung für 2060	DM 89,50
2065	Radio-Technik, Opto-Elektronik	DM 139,–
2069	Ergänzungspackung für 2065	DM 49,50
2070	Studio-Center	DM 179,–
2072	IC-Verstärkertechnik	DM 48,–
2075	Digital-Technik	DM 79,–
2079	Ergänzungspackung – Steckbausteine	DM 11,50
2087	Netzstrom-Schaltgerät	DM 69,50
2089	Ergänzungspackung – IC-Fassungen	DM 9,90
2095	Cassetten-Interface	DM 129,50
5964	Schwachstrom-Spezial-Relais	DM 14,90
2090	Mikro-Computer microtronic	DM 389,–

BEZUGS-MÖGLICHKEITEN

Beim Elektronik-Fachhandel, bei größeren Buchhandlungen oder direkt beim Franzis-Verlag, Karlstraße 37–41, 8000 München 2, Telefon (0 89) 51 17-2 39/-3 80.

Bei Bezug ab Verlag können Sie unter drei Möglichkeiten wählen, wobei den genannten Verkaufspreisen jeweils 3,70 DM Porto hinzuzurechnen sind:

1. Vorauszahlung auf unser Postscheckkonto München Nr. 813 75-809
2. Zusendung eines Schecks
3. Bestellung per Nachnahme (zuzüglich 1,70 DM Nachnahme-Gebühr)

Bitte denken Sie an genaue Bestell- und Absenderangaben.

Das electronic-Experimentier-System erhalten Sie in der Schweiz beim **Verlag Thali AG,** CH-6285 Hitzkirch und in Österreich beim **Fachbuch Center Erb,** Amerlingstraße 1, A-1061 Wien.

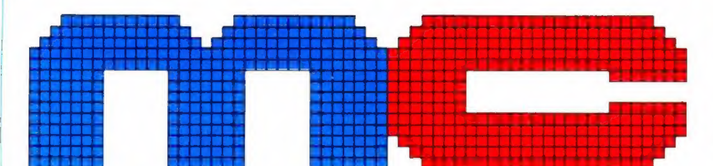


In Zusammenarbeit mit dem Elektronik-Magazin



Gerne übersenden wir Ihnen auf Anfrage einen ausführlichen Prospekt.
Franzis-Verlag, Karlstraße 37, 8000 München 2





macht Mikrocomputer anwendbar

Das finden Sie in mc:

- Einsatzmöglichkeiten der gängigen Hard- und Software.
- Laborerprobte Applikationen: Mikrocomputer dienen in vielen Bereichen zur Steuerung von Geräten und zur automatischen Erfassung und Auswertung von Meßergebnissen. mc sagt Ihnen wie, und hilft auch bei der Beseitigung von Interface-Problemen.
- Geprüfte Programme, für alle gängigen Rechner- und Prozessortypen.
- Für die Adaption vorhandener Programme auf andere Systeme gibt mc durch umfassende Dokumentation der rechnerinternen Betriebssoftware – die oft nicht einmal der Computer-Hersteller bietet – gezielte Anleitungen.
- Programmierhilfen: Als Software-Entwickler hilft Ihnen mc, Zeit und Geld zu sparen mit Standard-Routinen, Hilfsprogrammen, Softwarekniffen und Hinweisen zu Betriebssystem-Erweiterungen.
- Hardware-Tips: mc hilft Ihnen bei der Entwicklung und beim Bau individueller Hardware-Bausteine: Einplatinen-Mikrocomputer, Speicher-Erweiterungen, Interface-Schaltungen – kurz alles, was man eben doch selber machen muß.
- Software-Informationen: mc berichtet über neue Programmpakete, gibt Übersichten über Programmiersprachen und Betriebssysteme. In Grundlagenbeiträgen werden die logischen Zusammenhänge des Programmierens geschildert.
- Grundlagenbeiträge über Arbeitsweise und Eigenschaften von Mikrocomputern und Peripherie.
- Wie denken Computer? mc zeigt Ihnen, wie Ihr Tischcomputer intern „denkt". Das ermöglicht Ihnen, Ihr Gerät noch besser auszunutzen.
- Natürlich lesen Sie in mc auch wissenswerte Detail-Informationen aus der Mikrocomputer-Technik und ihren Randgebieten: Marktübersichten, Seminar-Termine, aktuelle Entwicklungen, Berichte über neue Hardware- und Softwareprodukte, Tests und Vergleiche von Computern und Programmiersprachen.
- Eine „Programm-Börse", in der Sie als Leser für eine minimale Schutzgebühr die Möglichkeit haben, Programme privat anzubieten.

mc erscheint monatlich. Das Einzelheft kostet 6,– DM im Inland, 6,50 DM im Ausland, das Jahresabonnement 60,– DM im Inland, 66,– DM im Ausland.

mc 4 April 1982 · 6 DM · 50 öS · 6.80 sfr
Die Mikrocomputer-Zeitschrift
Messen, Steuern, Regeln, Prüfen mit µCs
Komfortables Z80-Betriebssystem
IEC-Routinen für den 6502
Strichcode drucken

Ein Abonnement können Sie hier oder bei jeder Buchhandlung bestellen!

Bestellkarte

für ein Abonnement, Sonderheft oder Probeheft

Bitte senden Sie ab ____________ regelmäßig die Zeitschrift

☐ Elektronik
26 Hefte pro Jahr, Jahresabonnementspreis 115.20 DM im Inland, im Ausland 138.– DM.

☐ ELO
12 Hefte pro Jahr, Jahresabonnementspreis 39.60 DM im Inland, im Ausland 48.– DM.

☐ Funkschau
26 Hefte pro Jahr, Jahresabonnementspreis 96.– DM im Inland, im Ausland 118.80 DM.

☐ mc
12 Hefte pro Jahr, Jahresabonnementspreis 60.– DM im Inland, im Ausland 66.– DM.

Name/Vorname

Beruf

Straße

PLZ/Ort

Datum Unterschrift

In den genannten Abonnementspreisen sind sämtliche Nebenkosten, einschließlich Porto, enthalten. Bei verlagsseitiger Änderung muß die Berechnung aus gesetzlichen Gründen (Preisbindung!) zum neuen Preis erfolgen.
Die Kündigung ist jeweils 8 Wochen zum Kalenderjahresende möglich.
Die Abonnementsgebühr ist nach Erhalt der Rechnung fällig. Preise Stand 10/82. Falls Sie Abbuchung vom Konto wünschen, beachten Sie bitte die Rückseite. Diese Vereinbarung können Sie innerhalb einer Woche schriftlich widerrufen.

Multi-Mikrocomputer-Systeme

Bestellkarte

für ein Abonnement, Sonderheft oder Probeheft

Bitte senden Sie ab ____________ regelmäßig die Zeitschrift

☐ Elektronik
26 Hefte pro Jahr, Jahresabonnementspreis 115.20 DM im Inland, im Ausland 138.– DM.

☐ ELO
12 Hefte pro Jahr, Jahresabonnementspreis 39.60 DM im Inland, im Ausland 48.– DM.

☐ Funkschau
26 Hefte pro Jahr, Jahresabonnementspreis 96.– DM im Inland, im Ausland 118.80 DM.

☐ mc
12 Hefte pro Jahr, Jahresabonnementspreis 60.– DM im Inland, im Ausland 66.– DM.

Name/Vorname

Beruf

Straße

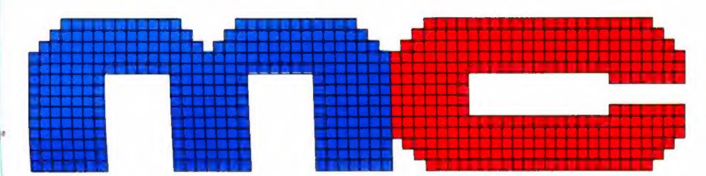
PLZ/Ort

Datum Unterschrift

In den genannten Abonnementspreisen sind sämtliche Nebenkosten, einschließlich Porto, enthalten. Bei verlagsseitiger Änderung muß die Berechnung aus gesetzlichen Gründen (Preisbindung!) zum neuen Preis erfolgen.
Die Kündigung ist jeweils 8 Wochen zum Kalenderjahresende möglich.
Die Abonnementsgebühr ist nach Erhalt der Rechnung fällig. Preise Stand 10/82. Falls Sie Abbuchung vom Konto wünschen, beachten Sie bitte die Rückseite. Diese Vereinbarung können Sie innerhalb einer Woche schriftlich widerrufen.

Multi-Mikrocomputer-Systeme





macht Mikrocomputer anwendbar

Das finden Sie in mc:

- Einsatzmöglichkeiten der gängigen Hard- und Software.
- Laborerprobte Applikationen: Mikrocomputer dienen in vielen Bereichen zur Steuerung von Geräten und zur automatischen Erfassung und Auswertung von Meßergebnissen. mc sagt Ihnen wie, und hilft auch bei der Beseitigung von Interface-Problemen.
- Geprüfte Programme, für alle gängigen Rechner- und Prozessortypen.
- Für die Adaption vorhandener Programme auf andere Systeme gibt mc durch umfassende Dokumentation der rechnerinternen Betriebssoftware – die oft nicht einmal der Computer-Hersteller bietet – gezielte Anleitungen.
- Programmierhilfen: Als Software-Entwickler hilft Ihnen mc, Zeit und Geld zu sparen mit Standard-Routinen, Hilfsprogrammen, Softwarekniffen und Hinweisen zu Betriebssystem-Erweiterungen.
- Hardware-Tips: mc hilft Ihnen bei der Entwicklung und beim Bau individueller Hardware-Bausteine: Einplatinen-Mikrocomputer, Speicher-Erweiterungen, Interface-Schaltungen – kurz alles, was man eben doch selber machen muß.
- Software-Informationen: mc berichtet über neue Programmpakete, gibt Übersichten über Programmiersprachen und Betriebssysteme. In Grundlagenbeiträgen werden die logischen Zusammenhänge des Programmierens geschildert.
- Grundlagenbeiträge über Arbeitsweise und Eigenschaften von Mikrocomputern und Peripherie.
- Wie denken Computer? mc zeigt Ihnen, wie Ihr Tischcomputer intern „denkt". Das ermöglicht Ihnen, Ihr Gerät noch besser auszunutzen.
- Natürlich lesen Sie in mc auch wissenswerte Detail-Informationen aus der Mikrocomputer-Technik und ihren Randgebieten: Marktübersichten, Seminar-Termine, aktuelle Entwicklungen, Berichte über neue Hardware- und Softwareprodukte, Tests und Vergleiche von Computern und Programmiersprachen.
- Eine „Programm-Börse", in der Sie als Leser für eine minimale Schutzgebühr die Möglichkeit haben, Programme privat anzubieten.

mc erscheint monatlich. Das Einzelheft kostet 6,– DM im Inland, 6,50 DM im Ausland, das Jahresabonnement 60,– DM im Inland, 66,– DM im Ausland.



Ein Abonnement können Sie hier oder bei jeder Buchhandlung bestellen!

Bestellkarte

für ein Abonnement, Sonderheft oder Probeheft

Bitte senden Sie ab ______________ regelmäßig die Zeitschrift

☐ **Elektronik**
26 Hefte pro Jahr, Jahresabonnementspreis 115.20 DM im Inland, im Ausland 138.– DM.

☐ 
12 Hefte pro Jahr, Jahresabonnementspreis 39.60 DM im Inland, im Ausland 48.– DM.

☐ 
26 Hefte pro Jahr, Jahresabonnementspreis 96.– DM im Inland, im Ausland 118.80 DM.

☐ 
12 Hefte pro Jahr, Jahresabonnementspreis 60.– DM im Inland, im Ausland 66.– DM.

Name/Vorname

Beruf

Straße

PLZ/Ort

Datum Unterschrift

In den genannten Abonnementspreisen sind sämtliche Nebenkosten, einschließlich Porto, enthalten. Bei verlagsseitiger Änderung muß die Berechnung aus gesetzlichen Gründen (Preisbindung!) zum neuen Preis erfolgen.
Die Kündigung ist jeweils 8 Wochen zum Kalenderjahresende möglich.
Die Abonnementsgebühr ist nach Erhalt der Rechnung fällig. Preise Stand 10/82. Falls Sie Abbuchung vom Konto wünschen, beachten Sie bitte die Rückseite. Diese Vereinbarung können Sie innerhalb einer Woche schriftlich widerrufen.

Multi-Mikrocomputer-Systeme

Bestellkarte

für ein Abonnement, Sonderheft oder Probeheft

Bitte senden Sie ab ______________ regelmäßig die Zeitschrift

☐ **Elektronik**
26 Hefte pro Jahr, Jahresabonnementspreis 115.20 DM im Inland, im Ausland 138.– DM.

☐ ELO
12 Hefte pro Jahr, Jahresabonnementspreis 39.60 DM im Inland, im Ausland 48.– DM.

☐ Funkschau
26 Hefte pro Jahr, Jahresabonnementspreis 96.– DM im Inland, im Ausland 118.80 DM.

☐ 

12 Hefte pro Jahr, Jahresabonnementspreis 60.– DM im Inland, im Ausland 66.– DM.

Name/Vorname

Beruf

Straße

PLZ/Ort

Datum Unterschrift

In den genannten Abonnementspreisen sind sämtliche Nebenkosten, einschließlich Porto, enthalten. Bei verlagsseitiger Änderung muß die Berechnung aus gesetzlichen Gründen (Preisbindung!) zum neuen Preis erfolgen.
Die Kündigung ist jeweils 8 Wochen zum Kalenderjahresende möglich.
Die Abonnementsgebühr ist nach Erhalt der Rechnung fällig. Preise Stand 10/82. Falls Sie Abbuchung vom Konto wünschen, beachten Sie bitte die Rückseite. Diese Vereinbarung können Sie innerhalb einer Woche schriftlich widerrufen.

Multi-Mikrocomputer-Systeme